ZÁPADOČESKÁ UNIVERZITA V PLZNI

FAKULTA ELEKTROTECHNICKÁ

KATEDRA APLIKOVANÉ ELEKTRONIKY A TELEKOMUNIKACÍ

DIPLOMOVÁ PRÁCE

Využití embedded modulů pro monitoring životních funkcí pacienta

Bc. Tomáš Cinert

Plzeň 2012

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2011/2012**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Tomáš CINERT**
Osobní číslo:	**E09N0148P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a aplikovaná informatika**
Název tématu:	**Využití embedded modulů pro monitoring životních funkcí pacienta**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

1. Prostudujte možnosti technologie Java ME v oblasti embedded modulů.
2. Vyberte vhodné metodiky a senzory pro monitoring životních funkcí pacienta.
3. Využijte embedded moduly s implementací platformy Java ME (SUNSpot, Cinterion apod.) s čidly polohy, teploty atd.
4. Sledujte u monitorované osoby pomocí senzorů srdeční tep, oxidaci kyslíkem, polohu (prudké změny polohy). V případě překročení stanovených mezí resp. náhlé změně polohy (pád, mdloba) systém vyhodnotí vznik zdravotního rizika. Systém tuto informaci předá (s využitím GSM modulu, WiFi apod.) lékaři. Vytvořte na platformě Java ME vhodnou aplikaci pro monitoring a rozpoznání uvedených stavů. Vytvořte mobilní zařízení umožňující monitorované osobě volný pohyb bez větších omezení.
5. Vyhodnoťte spolehlivost zařízení z hlediska použité technologie, vytvořeného programového vybavení a samotné platformy Java ME.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah pracovní zprávy: **30 - 40 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **Topley, K. : J2ME V kostce, Grada Publishing, Praha 2004**
2. **Mahmoud, Q. H.: Naučte se Java 2 Micro Edition, Grada Publishing, Praha 2002**
3. **Roedl, M.: Využití jazyka MicroJAVA na platformě mobilních zařízení, bakalářská práce ZČU 2006**
4. **Exnar, P.; Doležal, I.; Viková, M.; Jančík, P.: Monitorování životních funkcí - senzory teploty a vlhkosti, Výzkumné a vývojové projekty klastru 2006, str. 31 - 33, TU Liberec**

Vedoucí diplomové práce: **Ing. Petr Kropík, Ph.D.**
Katedra teoretické elektrotechniky

Konzultant diplomové práce: **Ing. Petr Kropík, Ph.D.**
Katedra teoretické elektrotechniky

Datum zadání diplomové práce: **18. října 2010**

Termín odevzdání diplomové práce: **11. května 2012**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 17. října 2011

Abstrakt

Tato diplomová práce se zabývá návrhem a realizací zařízení pro monitorování životních funkcí pacienta. Sleduje u osoby několik životních funkcí (tep, saturace kyslíkem a teplotu) a prudké změny polohy (pád). Zařízení tyto stavy kontinuálně vyhodnocuje a v případě ohrožení, předává tuto informaci dál (záchranná služba, lékař). Veškerá data jsou ukládána v lokální databázi a následně odesílána do vzdálené databáze. Naměřená data jsou tedy s jistým zpožděním dostupná i vzdáleně pro autorizované osoby.

Klíčová slova

AT příkazy, GSM modul, Java ME, monitorování životních funkcí, tep, saturace kyslíkem, teplota, pád, výjimka, databáze

Abstract

Goal of this thesis is to design and develop device for monitoring patient vital signs. The device measures several vital signs (pulse, oxygenation of a patient's hemoglobin and body temperature) and radical changes of position (downfall). The device evaluates these states and in an emergency, it sends this information further (rescue service, doctor). Data are stored in local database and they are also sent to remote database subsequently. Authorized person can see the measured data remotely with delay.

Key words

AT Commands, GSM module, Java ME, monitoring patient vital signs, pulse, oxygenation of a patient's hemoglobin, fall, exception, database

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni dne 4.5.2012 Jméno příjmení

……..

Poděkování

Tímto bych rád poděkoval vedoucímu diplomové práce Ing. Petru Kropíkovi, Ph.D. za cenné profesionální rady, připomínky a metodické vedení práce. Dále bych chtěl poděkovat své rodině a přátelům za podporu během celého studia.

Obsah

1. Úvod

Žijeme v moderní době, kde vývoj techniky jde skokem. Moderní technologie stále více prorůstá normálním životem a dostává se do každodenní praxe a prostupuje napříč všemi obory. Jedním z oborů, na který měla technologie velký vliv, je medicína – samozřejmě nezměnila základní snahu tohoto oboru a tou je zachraňovat lidské životy – pouze tuto snahu ulehčila. Dnes si již těžko představíme moderní medicínu bez pomoci techniky – jen pokud vezmeme v potaz přístroje, které používáme k zachraňování lidských životů (např. defiblirátor, kardiostimulátor), monitoring životních funkcí, operační přístroje (robotická ruka pro precizní operace, které lidská ruka není schopna vykonat) až po přístroje „menší" důležitosti např. v plastické chirurgii.

I přes tyto technické pokroky, kdy co se zdálo nemožné před deseti lety, je dnes naprosto běžné, jsou zde stále místa na které ani moderní medicína nestačí. Jednou z věcí je například předpověď, kdy dojde k nějaké zdravotní události (např. srdeční infarkt). Pokud bychom byli schopni toto určit, bylo by možné účinně zasáhnout. Dále stále existují nemoci, které nejsme schopni vyléčit – zatím můžeme pacientovi pouze ulehčit jeho trápení.

Cílem této práce je vytvořit zařízení, které samozřejmě není určeno k řešení výše popsaných problémů, ale k nápomoci řešení vzniklé situace. Zařízení je primárně určeno starším lidem a nemocným či postiženým lidem. Dochází poměrně často k situacím, kdy člověka postihne nějaká zdravotní komplikace (a to se nemusí pouze týkat výše popsané skupiny – může se jednat i naprosto zdravého jedince, sportovce, mladého člověka) a následkem této komplikace může dojít k druhotným zraněním (pád ze schodů, pád ze žebříku, apod.), které danou situaci zhorší. Často se lidé ocitnou v této situaci, kdy jsou sami a následně nejsou schopni se dovolat pomoci (z důvodu poruchy vědomí, zlomení nohy, …). V této situaci by mnohdy stačilo dostat k člověku laickou, nejlépe lékařskou pomoc. Často jsou rozhodující minuty, kdy pomoc přijde a tyto minuty rozhodují o životě či smrti. Samozřejmě musíme konstatovat, že část těchto situací by i při okamžité odborné lékařské péči skončila stejně.

Finální zařízení by mělo tyto situace detekovat a vyhodnotit a následně zareagovat – při vážné situaci okamžité volání záchranné služby, při méně závažných komplikacích volání lékaře, příbuzného nebo kamaráda, který dále rozhodne o dalším průběhu (odvezení na odborné vyšetření, podání léku, …). Zařízení je pouze navrženo k oznámení tohoto stavu – není navrženo k řešení vzniklé situace (myslím si, že zařízení, které by bylo schopno vyhodnotit a následně vyřešit všechny možné situace není v současné době reálně možné). Další možností tohoto zařízení je archivace dat a následné odeslání k vyhodnocení, takže pacient nemusí se zařízením k lékaři. Například mnoho srdečních arytmií vzniká ve spánku, kdy je pacient ani nemusí zaznamenat. Pomocí tohoto zařízení se dá monitorovat a zaznamenávat pacientovi životní funkce i během noci a proto může být doma a nemusí trávit noc v nemocnici. Zařízení automaticky měřené údaje odesílá do databáze.

Popsané zařízení může pomoci řešit i jinou situaci – zatím byla popsána situace s prvotní zdravotní příčinou, ale může nastat i jiná situace a to je úraz – pád ze žebříku, pád ze schodů atd., který nemusí být způsoben zdravotním problémem – i tyto situace zařízení umí signalizovat a následně vyhodnotit.

2. AT příkazy

2.1 Historie

Počátky rozvoje jsou spojeny s rozvojem sítí POTS/PSTN (the Plain Old Telephone System/Public Switched Telephone Network) – jedná se o analogovou telefonní síť v USA (POTS) a Evropě (PSTN) v počátku 1980 [1]. Nejrozšířenější variantou byl modem s manuální volbou, kdy uživatel musel „vytočit" číslo a čekat na spojení a poté spojení také ukončit. V této situaci přišla společnost Hayes s modemem Smartmodem 1200 a chvíli poté typ Smartmodem 2400[1]. Hlavní rozdíl oproti klasickým byl v možnosti automatické volby. Navíc tyto dva modemy byly řiditelné stejnými příkazy – v té době to nebylo obvyklé, výrobci si nelámali hlavu s kompatibilitou s předchozími typy. Řešení společnosti Hayes bylo výjimečné svým principem – modem sám o sobě mohl přepínat mezi dvěma módy:

- **`Datový mód`**
- **`Příkazový mód`**

Toto řešení mělo velký úspěch, proto několik výrobců začalo nabízet „Hayes kompatibilní" modemy. Společnost Hayes na ně podala soudní žalobu - skupina výrobců začala používat název „AT commands kompatibilní." [2]

2.2 Popis

AT příkazy se dají rozdělit na dvě skupiny:

- **`Základní`** – *jedná o elementární příkazy v rozsahu standardních modemů. Část příkazů je pouze informativní pro ověření vlastností. Existuje podskupina speciálních příkazů doplněných znaky: '&', '\', '%' a '*'.*
- **`Rozšířené`** – *pro provádění nastavení a akcí. Obsahují i příkazy specializované na různé nadstavbové činnosti (GSM apod.). Největší část z nich používá prefix '+' následován třemi znaky.*

[3]

2.3 Použití

Obecně jsou AT příkazy pro komunikaci mezi DTE (Data Terminal Equipment) a modemem (nebo obecně DCE - Data Communications Equipment), kdy modem je v příkazovém módu. Každý příkaz má tři části:

- **`Prefix`** *(např. '&' nebo '+')*
- **`Tělo`**
- **`Terminátor`** *('<CR><LF>' znak)* [4]

[1] Číslo značí přenosovou rychlost v baud/s.

Všechny příkazy musí začít sekvencí „AT“ a příkaz musí být ukončen terminátorem. Jakékoliv znaky před touto sekvencí jsou ignorovány (kromě speciálního příkazu „A/“ – zopakování posledního příkazu). Základní použití definuje tři možnosti:

1. **`Zjištění parametru`** – *na konec těla příkazu vložíme ‘?’, např. AT+CPIN?*
2. **`Nastavení parametru`** – *na konec těla příkazu vložíme ‘=’ např. AT+CPIN=1234 , pokud nezadáme hodnotu je defaultně nastavena 0.*
3. **`Možné parametry`** – *kombinace předcházejících „=?“, návratová hodnota je seznam možných parametrů, např. AT+CPIN=?*

2.4 Seznam

Seznam všech AT příkazů by zabral několik stránek, protože příkazů je mnoho. U různých zařízení se mohou lišit – v zásadě nejde o rozdíly, spíše o způsob implementace daného příkazu, u některých příkazů mohou být odlišné parametry v závislosti na typu zařízení apod. Proto zde bude pouze výpis použitých AT příkazů a jejich možných parametrů (v některý případech ve zkrácené verzi). Popsané AT příkazy jsou pro použité zařízení Cinterion TC65, takže u jiného typu se mohou lišit. Celý seznam použitelných příkazů lze nalézt v *Datasheet TC65 AT Commands* [5] – odtud jsou také čerpány informace o jednotlivých příkazech.

`AT (ATtention)`

Dlouhý formát	Krátký formát	Popis
OK	0	Modem je ve funkčním stavu. Otestování odezvy.
CONNECT	1	Spojení sestaveno.
RING	2	Zjištěn příchozí hovor.
ERROR	4	Došlo k chybě.
BUSY	7	Detekován tón obsazeno.
DATA	35	Datové spojení navázáno.
CONNECT 110	24	Přenosová rychlost 110 bps.

Tabulka 1: AT

Tabulka není celá. Je vybráno pouze několik možných odpovědí. Jak si lze představit z posledního řádku, odpovědí typu „CONNECT xxxx“ je možných velké množství dle nastavené přenosové rychlosti. Další možné odpovědi jako např. navázání faxového spojení není pro tuto práci nutno uvádět.

2.4.1 Funkční příkazy

AT&F

Nastavení všech parametrů do výchozího (továrního) nastavení. Po provedení modem potvrzuje odpovědí „OK“. Tento příkaz nemaže uživatelem definovaný profil, který lze vyvolat příkazem ***ATZ*** (viz dále).

AT+W

Nastavení současných parametrů jako uživatelsky definovaný profil. Tento profil bude automaticky obnoven po zapnutí, dokud nedojde k obnovení továrního nastavení či jeho změně.

ATZ

Nastavení uživatelského profilu, který byl definován pomocí ***AT&W.*** Nejdříve je zařízení uvedeno do továrního nastavení a až poté do požadovaného profilu (pokud je dostupný). Je doporučeno počkat 300 ms před dalším příkazem.

AT+CFUN

Nastavení funkce zařízení. Tento příkaz může být použit k resetování zařízení, k výběru jednoho z úsporných módů nebo k návratu do plné funkčnosti. Jsou dostupné dva typy úsporných módů:

- **NON-CYCLIC SLEEP** (*necyklický úsporný mód*) – *permanentně blokuje sériové rozhraní.*
- **CYCLIC SLEEP** (*cyklický úsporný mód) – je možné vzbudit zařízení externě a navíc je možná větší funkčnost zařízení.*

Možné parametry	Popis
0	NON-CYCLIC SLEEP
1	Plně funkční
7	CYCLIC SLEEP
9	CYCLIC SLEEP

Tabulka 2: AT+CFUN

AT+CMEE

Nastavení formátu chybové zprávy. Možné formáty:

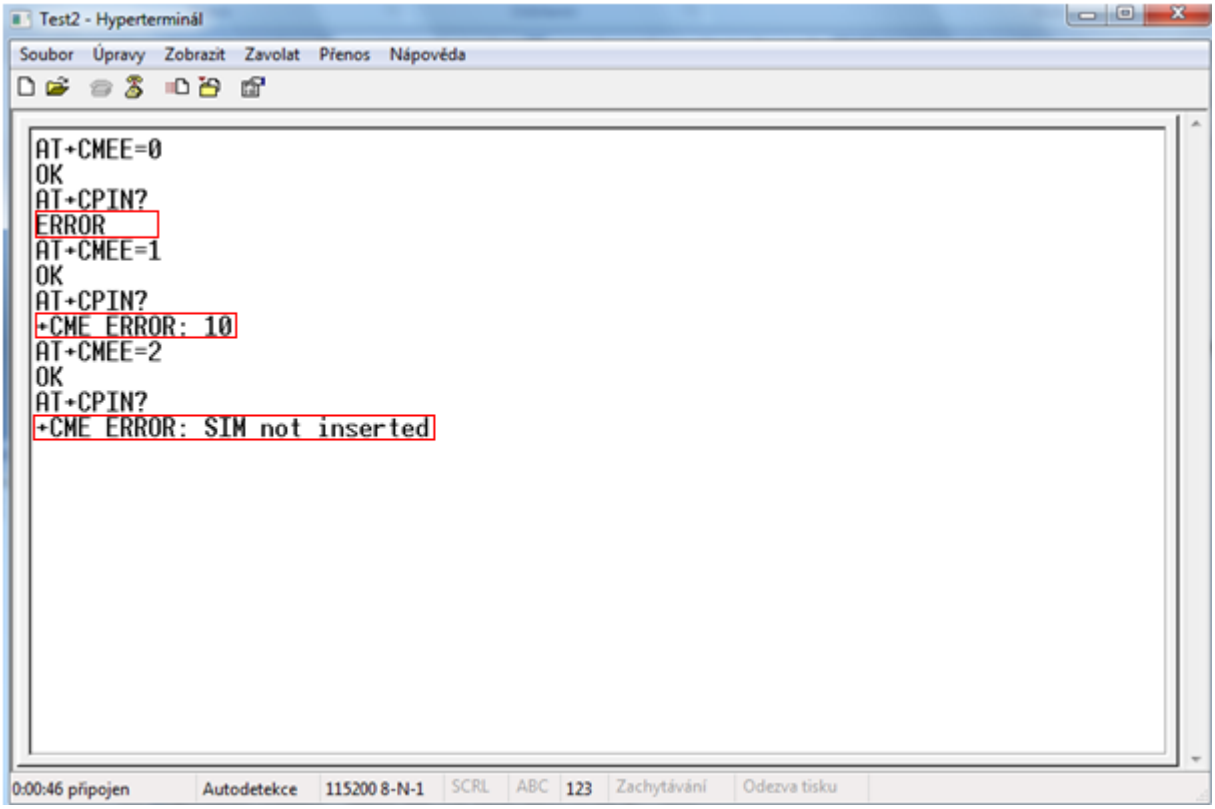
- **Vypnutí chybového hlášení** – *resp. pouhý výpis **ERROR** při chybě*
- **Číslo chyby**

- `Popis chyby`
  - *Obecný chybový kód*
  - *Obecný chybový kód SIEMENS*
  - *Chybový kód spojený s GPRS*
  - *Chybový kód spojený s SMS*

Možné parametry	Popis
0	Vypnutí
1	Číslo chyby
2	Popis chyby

Tabulka 3: AT+CMEE parametry

Na následujícím obrázku lze vidět rozdíl ve výpisech chyb. Jde o situaci, kdy není vložena SIM karta a je odeslán AT příkaz pro zadání/zjištění stavu bezpečnostního kódu PIN.



Obr. 1: Nastavení výpisu chyb

Jak je vidět prvně je nastaveno „vypnutí" výpisu, takže vidíme pouze výpis ERROR bez jakékoliv specifikace. Dále je nastaven druhý mód „číslo chyby" a vidíme, že součástí chybového výpisu je číslo chyby, podle kterého potom v příslušné dokumentaci můžeme najít problém. A v posledním případě je nastaven výpis s popisem chyby a vidíme, že `SIM not inserted` – SIM karta není vložena.

AT^SIND

Rozšířená kontrola indikátorů. Slouží jako náhrada staršího ***AT+CIND*** oproti němu nabízí větší flexibilitu a možnost kontrolovat vice indikátorů. Rozsah parametrů (popisu indikátorů) je od audia po síťovou identifikaci až po status SIM karty.

AT^SBV

Tento příkaz monitoruje napětí zdroje tohoto modulu. Je vrácena průměrná hodnota z několika měření v mV.

2.4.3 Síťové příkazy

AT+CREG

Slouží k ověření registrace do sítě. Status registrace:

- `0` – *nezaregistrováno a mobilní zařízení nehledá síť. Většinou se jedná o dočasný stav. Pokud ne, může to indikovat jeden z následujících problémů:*
    - *Automatický mód*
        - *Není vložena SIM karta.*
        - *Nezadán PIN.*
        - *Nenalezen žádný záznam operátorů na SIM kartě.*
    - *Manuální mód*
        - *Domovský operátor není povolen.*
        - *Přihlášení v dané lokalitě není dovoleno.*
        - *Není povolen roaming.*

  Uživatelský zásah je potřeba k vyřešení problému. Tísňové volání je možné (za předpokladu, že je dostupná nějaká síť).
- `1` – *Přihlášen do domovské sítě.*
- `2` – *Nepřihlášen. Mobilní zařízení hledá síť. Pokud ani po minutě nedojde k přihlášení do sítě, jedná se patrně o jeden z následujících problémů:*
    - *Žádná síť v dosahu nebo nedostačující Rx[2] úroveň.*
    - *Mobilní zařízení nemá právo přihlášení do dostupných sítí.*
    - *Povolené sítě jsou v dosahu, ale přihlášení stejně selže z důvodu:*
        - *Domovská síť není povolena.*
        - *Přihlášení v dané lokalitě není povoleno.*
        - *Není povolen roaming.*

  Pokud alespoň jedna síť je dostupná, lze zavolat na tísňovou linku.
- `3` – *Přihlášení zamítnuto. K tomu může dojít z několika důvodů:*

[2] Kvalita přijímaného signálu

  - *Neznámé IMSI[3] v HLR[4].*
  - *Nelegální mobilní stanice.*
  - *Nelegální mobilní zařízení.*
- **4** – *Nepoužito.*
- **5** – *Přihlášen v zahraniční síti (roaming).*

2.4.4 Příkazy pro SMS (Short Message Service)

AT+CMGF

Zvolení formátu SMS.

Možné parametry	Popis
0	PDU mód (binární)
1	Textový mód (defaultní)

Tabulka 4: AT+CMGF

AT+CMGC

Příkaz pro poslání SMS. Formát

AT+CMGC="číslo***"<CR>***Text zprávy***<CTRL-Z>***

2.4.5 Příkazy pro GPIO

AT^SSPI

Příkaz pro zahájení a konfiguraci SPI komunikace. Použití lze vidět na příkladu

```
ATsend("AT^SSPI=1020,0000,0000,0000 <1000010>\r");
```

1020

- **1** – *volba SPI*
- **0** – *komunikace přes interní konektor*
- **2** – *rychlost přenosu – 500Kb/s (možnost nastavit od 100kb/s až do 6.5Mb/s)*
- **0** – *ASCII kódování*

0000

- *Zpoždění po každém zapsaném bajtu. Pro SPI nepoužitelné.*

[3] International Mobile Subscriber Identity – unikátní číslo přidělené operátorem pro SIM kartu
[4] HLR – Home Location Register – domovský lokační registr

0000

- *Zpoždění po každém přečteném bajtu. Pro SPI nepoužitelné.*

0000

- **0** *– SPI mód. Je možné vybírat ze 4 módů. Vzorkování (náběžná, doběžná hrana) a zda hodiny jsou v klidu ve vysoké nebo nízké úrovni.*
- **0** *– CS (Chip Select). Zde není možnost volby.*
- **0** *– Uspořádání bajtů – opět bez možnosti volby.*
- **0** *– MSB bit první.*

<1000010> Zpráva pro přenos

- **<** *- Speciální znak oznamující počátek zprávy.*
- **1** *– Identifikátor. TC65 je napevno master a tudíž má přiřazen identifikátor 0, proto identifikátory pro další zařízení (slave) začíná od 1.*
- **00** *– offset pro čtení – od kterého bitu se má začít číst. V našem případě od začátku.*
- **0010** *– požadavek o zaslání 16 bajtů.*
- **>** *- ukončovací znak.*

Data jsou přijaty pomocí URC (*Unsolicited Result Code*).

4. Cinterion TC65

Tato kapitola se zabývá popisem modulu Cinterion TC65. Jedná se o GSM modul, který byl původně vyvinutý firmou Siemens. Veškeré informace jsou čerpány z *Datasheet TC65 Terminal Hardware Description* [6].

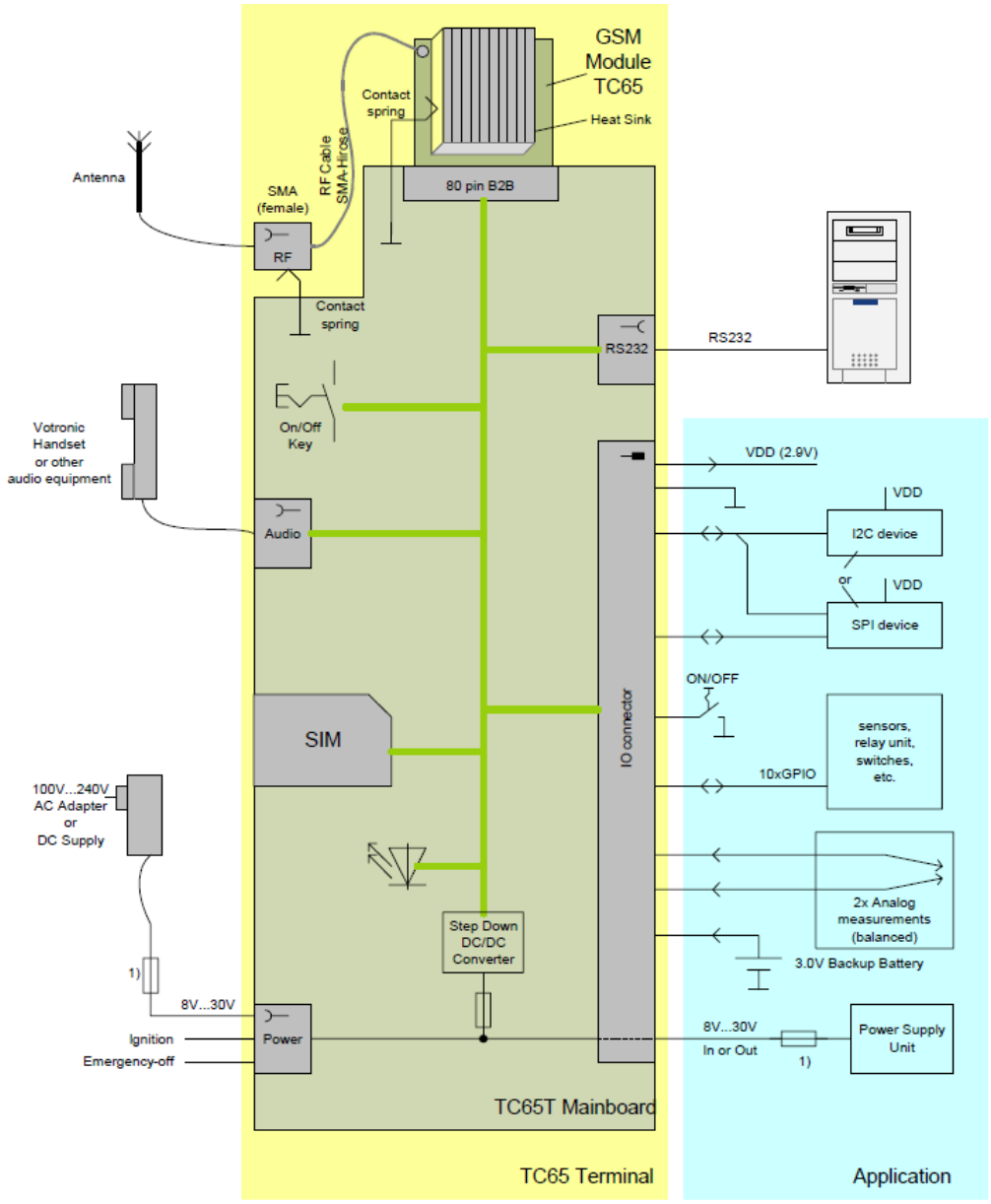
4.1 Základní popis

Vlastnosti	**Implementace**
Frekvenční pásmo	Quad band: GSM 850/900/1800/1900 MHz
GSM třída	Small MS (*Malá mobilní stanice*)
Napájecí zdroj	8 až 30V
Provozní teplota	-30°C až +65°C
Rozměry	130mm x 90mm x 38mm
Váha	190g
Datové přenosy	GPRS (Třída 12), CSD
Komunikace	GPIO
Fax	Skupina 3, třída 1

Tabulka 5: Základní vlastnosti

4.2 Schéma

Je přiloženo schéma pro nastínění vlastností modulu. Převzato z [6].



Obr. 2: **Blokové schéma TC65**

4.3 Popis hardwaru

Na blokovém schématu jsou vidět nejdůležitější části modulu. Nebudeme se zde zabývat detailním popisem jednotlivých částí, jelikož vše potřebné lze najít v *TC65_Terminal_Hardware_Description* [6].

4.3.1 IO rozhraní

Obsahuje tyto rozhraní:

- **Programovatelné GPIO** *(General Purpose Input Output)*
- **I²C sběrnice** *(Inter-Integrated Circuit)*
- **SPI** *(Serial Peripheal Interface)*
- **Dva analogové vstupy** *(vyvážené)*
- **Napájení**
- **Záložní napájení**
- **VDD napájení**
- **Přepínač On/Off**
- **Čítač impulsů**

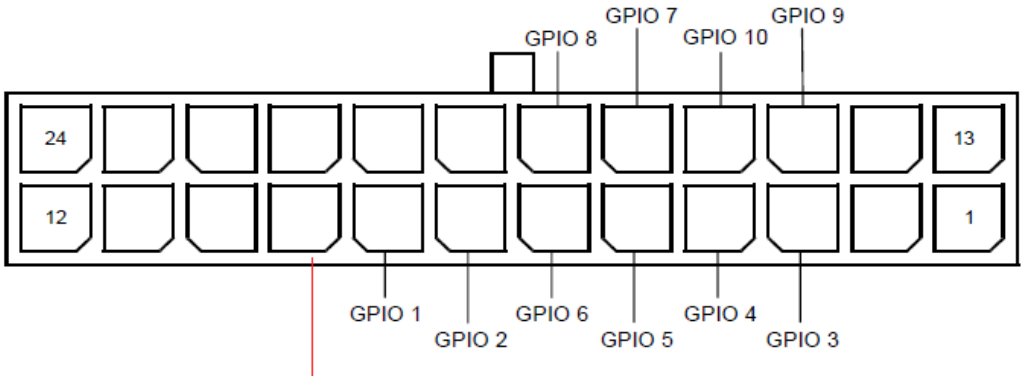
PIN	Jméno	I/O	Popis
1	I2CCLK_SPICLK	O	I2C nebo SPI hodiny
2	I 2CDAT_SPIDO	I/O	I2C data nebo SPI data ven
3	GPIO3	I/O	Programovatelné GPIO
4	GPIO4	I/O	Programovatelné GPIO
5	GPIO5	I/O	Programovatelné GPIO
6	GPIO6	I/O	Programovatelné GPIO
7	GPIO2	I/O	Programovatelné GPIO
8	GPIO1	I/O	Programovatelné GPIO
9	ADC2_IN_P	I	Vyvážený analogový vstup kladný
10	ADC2_IN_N	I	Vyvážený analogový vstup záporný
11	BackUp	I	Záložní baterie
12	GND	-	Zem
13	SPICS	O	SPI výběr
14	SPIDI	I	SPID data dovnitř
15	GPIO9	I/O	Programovatelné GPIO
16	GPIO10	I/O	Programovatelné GPIO/Čítač impulzů
17	GPIO7	I/O	Programovatelné GPIO
18	GPIO8	I/O	Programovatelné GPIO
19	VDD	O	Signálové napájení
20	ONOFF	I	Zapínání
21	ADC1_IN_P	I	Vyvážený analogový vstup kladný
22	ADC1_IN_N	I	Vyvážený analogový vstup záporný

23	GND	-	Zem (signálová)
24	Power	I	Napájení

Tabulka 6: Piny IO konektoru

GPIO

Modul poskytuje 10 GPIO pinů. Všechny jsou chráněny proti elektrostatickému výboji. Odpor 100 Ω je zapojen v sérii – to zabraňuje zkratům a je to velmi důležité hlavně v první fázi, kdy JAVA aplikace není úplně implementována. Volba, zda jde o výstup či vstup, je závislá na konfiguraci pomocí AT příkazů. Pin číslo 10 lze použít jako čítač impulzů v rozsahu 0 až 1000 za sekundu.



Obr. 3: **IO konektor s označenými GPIO piny**

Po startu modulu jsou všechny GPIO piny ve vysoko impedančním stavu. Proto je doporučeno zapojit do všech GPIO pinů, které chceme použít jako výstupy, pull-up nebo pull-down rezistory – to zabrání „plavání“ pinu.

4.3.3 RTC záložní napájení

RTC (Real Time Clock) je napájen z odděleného zdroje uvnitř modulu. Pokud není připojen záložní zdroj, kondenzátor 100µF je použit pro napájení (zhruba na 6s).

4.3.4 Použití

GSM modul je použit jako řídící blok (více viz kapitola 9. Funkce). Vyhodnocuje přijatá data od měřící desky a rozhoduje o alarmu. Jak již bylo zmíněno, tento modul původně vyvinula společnost Siemens. Krátce poté byla divize této společnosti zabývající se těmito moduly koupena společností Cinterion. Bohužel tato společnost se zaměřila na novou produktovou řadu bez zpětné vazby se staršími. Proto vývojový software zůstal v podobě, jak ho vyvinula společnost Siemens. Každý software prochází určitým vývojovým cyklem, kdy jsou odchytávány problémy a řešeny, bohužel v tomto případě se tento cyklus zastavil před koncem (více o tomto v kapitole GSM modul TC65).

Modul je spojen s měřící deskou. Způsoby jsou dva:

- **Komunikační** – *RS232, SPI.*
- **Signalizační** – *data přijaty v pořádku, změna komunikace, pin pro signalizaci alarmu.*

5. JAVA

Vývoj tohoto programovacího jazyka začal v roce 1991 ve společnosti Sun Microsystems. Původním záměrem bylo vytvořit programovací jazyk pro spotřební zařízení (setop boxy, „chytré" dálkové ovladače, …). Tato zařízení měla omezený výpočetní výkon, proto cílem bylo vytvořit efektivní kód a také přenositelný (nezávislý na platformě, jelikož různé systémy měli různé vybavení). Nejdříve vývojáři používali C++, ale z důvodů různých problému, se rozhodli vytvořit zcela vlastní jazyk. Základy architektury nového jazyka čerpaly z různých jazyků – Eiffel, SmallTalk, Objective C a Cesar/Mesa. Tým těchto vývojářů se jmenoval „Green Team" a programovací jazyk byl nazván Oak. Později byl přejmenován na JAVA. První demonstrací nového jazyka byl dálkový ovladač domácího zábavního systému, ale tento nápad neslavil úspěch. Proto došlo k změně cílové platformy na webové aplikace – toto rozhodnutím můžeme nazvat „pravým" začátkem tohoto programovacího jazyka. Jazyk byl vyvíjen s mottem: „Napiš jednou, spusť všude". V dnešní době je JAVA všude – od mobilních telefonů přes navigační systémy až po firemní aplikace [7], [8].

5.1 Vlastnosti

Cílem vývoje bylo vytvořit jazyk, který bude bezpečný, výkonný, robustní a mutliplatformní. Proto byl vytvořen systém kompilace kódu na „prostředním" zařízení – tzv. virtuální stroj. Výstupem je bajtový kód, což je strojový jazyk *Java Virtual Machine* (JVM). To zjednodušuje práci pro programátora, jelikož program napsaný např. na platformě Windows může bez jakékoliv změny spustit např. na Linuxu nebo i jiné hardwarové platformě. Je úkolem JVM kód upravit pro příslušnou platformu. Zjednodušeně se dá říci, že JVM je interpret pro danou platformu. [9].

5.2 Java 2 Microedition

Tato kapitola popisuje použití programovací jazyk Java 2 Microedition. Informace v této kapitole (a podkapitolách) byly čerpány z [10], [11], [12], [13].

Možná se zdá, že není potřeba této edice Javy, jelikož dnešní elektronická zařízení (mobilní telefony, PDA, …) mají výkonnější CPU a více paměti než počítače, na kterých běžela první verze Javy. Ale klíčová vlastnost Javy ME je ve velikosti a výkonu. Potřeba této edice vychází hlavně z porovnání normálního zařízení (např. PC) a mobilního zařízení (např. „chytré" telefony) a to několika klíčových vlastností:

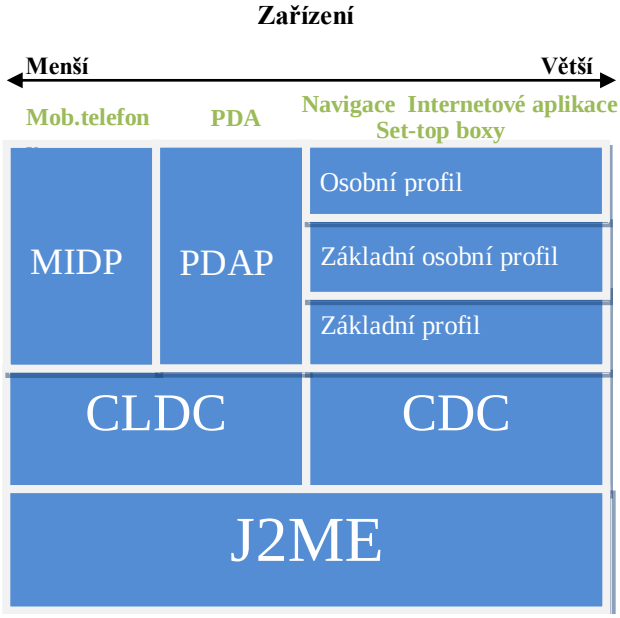
- **Velikost**
- **Spotřeba**
- **Méně výkonný procesor**
- **Méně paměti**

Obecně můžeme říct, že u osobního počítače se nemusíme starat o spotřebu, jelikož je stále zapojen do elektrické sítě, také se nemusíme „starat“, zda náš program zabírá o několik kB paměti více a požaduje o trochu vyšší výkon. Naproti tomu u mobilního zařízení, které obvykle běží na baterie, již několik kB navíc může hrát podstatnou roli. Cílem je umožnit psát flexibilní přenositelné programy pro mobilní zařízení s trvalým či částečným přístupem k síti. Vývoj J2ME je zajištěn projektem Java Community Process (JCP), který umožňuje komukoliv připojit se k vývoji této platformy.

Z obecného pohledu J2ME definuje následující komponenty:

- **Sérii virtuálních strojů** - *z nichž každý se uplatňuje u jiného typu malých zařízení s odlišnými nároky.*
- **Skupinu knihoven a API** - *spustitelné na každém virtuálním stroji (konfigurace a profily).*
- **Nástroje pro vývoj a nastavení zařízení.**

První dvě komponenty tvoří pracovní prostředí J2ME. I zde je JVM, která tvoří centrum prostředí. Nad ním se nachází specifická konfigurace, která se skládá z programovacích knihoven - ty zajišťují základní funkce vycházející z požadavků na zdroje daného zařízení. Vrchol tvoří jeden či více J2ME profilů.



Obr. 4: Architektura J2ME

5.3.1Konfigurace

Různá mobilní zařízení mají různou formu, funkci a vlastnosti, ale většinou používají podobné procesory a podobné množství paměti. Konfigurace tedy rozdělují zařízení do skupin na základě těchto dvou kritérií. Další doplňkové vlastnosti jsou podporované rysy programovacího jazyka Java, podporované rysy JVM a podpora knihoven. Existují dvě konfigurace:

- **CDC** *(Connected Device Configuration)*
- **CLDC** *(Connected Limited Device Configuration)*

CLDC

Tato konfigurace je určena pro méně výkonná zařízení, jak plyne z názvu „Limited" – omezená. Takže se jedná o zařízení s omezeným (nebo žádným) displejem, s omezenou pamětí, omezenou rychlostí CPU, omezenými vstupy, atd. Mnoho Java tříd zde nejsou dostupné (hlavně v oblasti kolekcí). Požadavky [14]:

- **Paměť**- *minimálně 160 kB paměti pro Java platformu.*
- **Procesor** - *8-35MHz.*
- **Napájení** - *limitované (většinou baterie)*
- **Připojení k síti** *(s možnou limitací šířky pásma)*
- **Rozhraní** - *možnost uživatelského rozhraní až po žádné*

Tato implementace je založená na malém JVM nazvaném KVM(*Kilo VM*). V JVM je implementována verifikace bajtového kódu – tato činnost je „náročnější" na paměť a procesor, proto zde je to sdíleno mezi vývojářem a KVM. Metoda Sandbox je zde také implementována a tato funkce je zde vyzdvihnuta – jednotlivé aplikace jsou odděleny v samostatných sandboxech.

Další klíčová vlastnost je *Generic Connection Framework* (GFC), která definuje hierarchii, která zobecňuje spojení (včetně TCP/IP, UDP a sériového portu).

První verze byla vydána v roce 2000 (CDLC 1.0). Poté byla založena skupina nazvaná „Expert Group", která rozhodovala o nové podobě této konfiguraci. Hlavním cílem nové specifikace bylo učinit tuto konfiguraci vyhovující pro Java specifikaci a JVM předěláním několika vlastností. Knihovny této konfigurace nebudou významně rozšířeny, z důvodu dodržení striktní specifikace (s ohledem na paměť). Další teoretickou možností bylo přidání minimálního bezpečnostního manažera a/nebo podpora odebrání tříd. Hlavní požadavky na novou verzi [14]:

- **Chybí podpora plovoucí desetinné čárky**

- **Chybí finalizace** – *neobsahuje* **Object.finalize()**
- **Omezené možnosti zpracování chyb** – *definuje pouze tři třídy pro zpracování chyb:*
  - **java.lang.Error**
  - **java.lang.OutOfMemoryError**
  - **java.lang.VirtualMachineError**
- **Chybí Java Native Interface**[5] – *z důvodu zabezpečení a omezené paměti*
- **Chybí uživatelem definované zavaděče tříd** – *zavaděč tříd je vestavěný. Důvod bezpečnost.*
- **Podpora reflexe není k dispozici**
- **Chybí skupiny vláken nebo podprocesy typu daemon**
- **Chybí „slabé" odkazy**[6]

Finálně v roce 2003 byla vydána nová verze CLCD1.1. Do této verze byla zapracována většina požadavků (nedostatků) předcházející (např. objekt finalizace není ani v nové verzi). Byly přidány další vlastnosti (např. pro práci s řetězci). Tato verze je stále aktuální.

5.3.2 Virtuální stroje

Virtuální stroj pro CLDC se nazývá KVM (Kilo Virtual Machine) a pro CDC CVM.

KVM

Je určeno pro malé zařízení dle specifikace CLDC. Je určen pro zařízení s přibližně 128 K dostupné paměti. Zdrojový kód byl napsán v jazyce C.

Je zde verifikace tříd – jedná se schopnost odmítnout neplatné *class* soubory během práce programu. Toto je součástí standardního JVM. Tato činnost je ale „náročnější" na paměť, proto byla přesunuta na počítač, kde se tyto soubory kompilují nebo na server, odkud se aplikace nahrává. Tento krok (verifikace mimo zařízení) se nazývá předběžné ověření (preverifikace).

[5] Toto rozhraní umožňuje integraci jiného programovacího jazyka (C, C++, atd.) do Java kódu
[6] Není dostatečně silná k donucení objektu k zůstání v paměti. Takže garbage collector rozhodne o znovu užití.

5.3.3 Profily

Je to sada programových rozhraní (API) tvořící nadstavbu konfigurace. Profil nabízí programu přístup k vlastnostem specifickým pro dané zařízení. Bude popsán pouze profil MIDP a jeho podtřída IMP. Informace čerpány z [15], [16].

MIDP (Mobile *Information Device Profile*)

Definuje otevřené prostředí pro vývoj aplikace pro mobilní informační zařízení (Mobile Information Device). Tato specifikace byla vytvořena skupinou více než 50 firem včetně vedoucích firem v oboru a výrobci softwaru pro mobilní zařízení (Motorola, Nokia, Siemens American Online – AOL,Sony, Oracle, Mitsubishi, …). Tento profil nedefinuje jen rozhraní a třídy, ale také popisuje, jak je aplikace instalována do zařízení. Aplikace jsou doplněny tzv. popisovačem aplikace, který obsahuje informace o aplikaci (informace o autorovi, jméno aplikace a velikost). V rámci MIDP je také definováno „povolení“ – pro nastavení přístupu k jednotlivým API.

IMP

Informační modulový profil byl představen společnostmi Siemens a Nokia v roce 2003. Je určen pro zařízení bez displeje. Vznikl jako podtřída MIDP 1.0 – neobsahoval API pro uživatelské rozhraní (**`javax.microedition.lcdui`**). Z toho lze vyvodit jeho určení – modemy, domácí elektronická zařízení a zařízení pro průmysl. Určen též pro zařízení s primitivním uživatelským rozhraním (LED diody, tlačítka). Aplikace napsané pod tímto profilem jsou v některých publikacích nazývány IMlety [17] – z důvodu odlišení od MIDletů, ale v podstatě architektura obou je velice podobná.

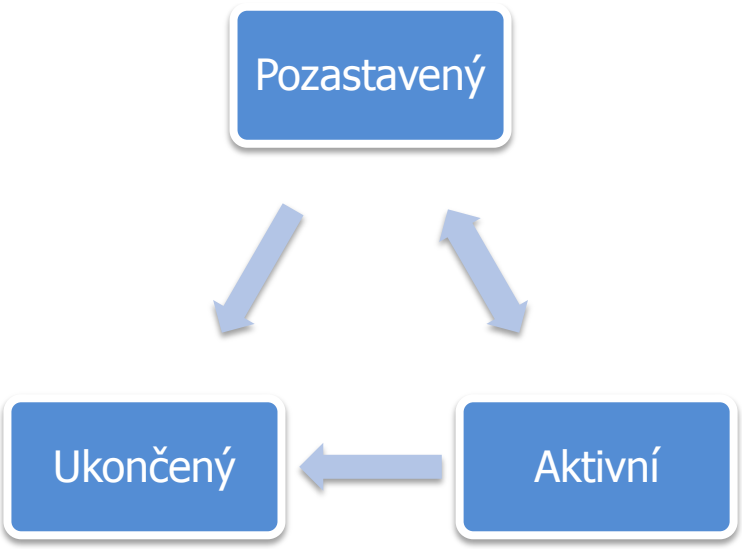
IMP-NG

Je odvezen od MIDP 2.0 opět bez podpory uživatelského rozhraní. Základní vlastnosti:

- **`Zpětná kompatibilita s IMP`**
- **`Udržet malé jádro`**
- **`Implementace veškerých užitečných funkcí z MIDP 2.0`**
- **`OTA`**
- **`„Push“ architektura`**

MIDlet

MIDlet je aplikace, která pracuje na MIDP zařízeních. Všechny MIDlety jsou potomkem třídy ***javax.microedition.midlet.MIDlet***. MIDlet pracuje, je v řízeném prostředí, a proto musí disponovat jistými funkcemi umožňujícími *manažeru aplikací* (který má na starosti instalace a spouštění MIDletu) kontrolovat chování aplikace. „Životní cyklus" MIDletu:



Obr. 6: Životní cyklus MIDlet

Jak je vidět z obrázku, MIDlet se může nacházet ve třech stavech:

- **`Pozastavený`** *(paused)*
- **`AKtivní`** *(actived)*
- **`Ukončený`** *(destroyed)*

MIDlet prochází těmito stavy následovně:

1. Když je MIDlet „spuštěn", instance je vytvořena a volán konstruktor MIDletu. Nachází se v přerušeném stavu.
2. MIDlet je v aktivním stavu poté co manažer aplikace zavolá metodu startApp().
3. V aktivní fázi může být MIDlet přerušen voláním funkce ***`pauseApp()`*** – tuto metodu volá manažer aplikace. MIDlet se sám může umístit do přerušeného stavu voláním metody ***`notifyPaused()`***
4. V kterékoliv fázi může být MIDlet přesunut do fáze zrušený voláním metody ***`destroyApp()`***. MIDlet se do zrušeného stavu může přesunout sám voláním metody ***`notifyDestroyed()`***.

Manažer aplikace (AMS – Application Management System) se stará o instalaci a spouštění aplikace. Po spuštění MIDletu, se nachází v přerušeném stavu. Jen AMS ho může

převést do aktivního stavu pomocí volání metody **`startApp()`**. AMS může kdykoliv MIDlet přerušit a převést ho do pozastaveného stavu nebo rovnou do zrušeného stavu. Situaci si můžeme nejlépe představit na mobilním telefonu – při příchozím hovoru je aplikace pozastavena a zdroje jsou uvolněny pro správu příchozí události.

5.4 Java EE

Enterprise Edition – sada specifikací pro API a distribuovanou architekturu PC plus definice pro zabalení komponent. Tato část Javy bude popsána jen ve zkratce – je použita pouze okrajově v této práci. Informace byly čerpány z [18], [15].

Je to kolekce standardizovaných komponentů, kontejnerů a služeb pro vytváření a spouštění distribuovaných aplikací. Často se jedná o software neběžící pouze na jednom PC, proto jsou zde nutné nástroje na rozdělení programu na několik částí. Oproti standardní verzi (Java SE) přidává rozhraní a knihovny. Pro vytváření grafického rozhraní lze použít komponenty z Java SE (Swing, AWT) nebo zcela nové rozhraní speciálně navržené pro Java EE – JavaServer Faces(JSF).

5.4.1 Základní vlastnosti

Několik základních pojmů:

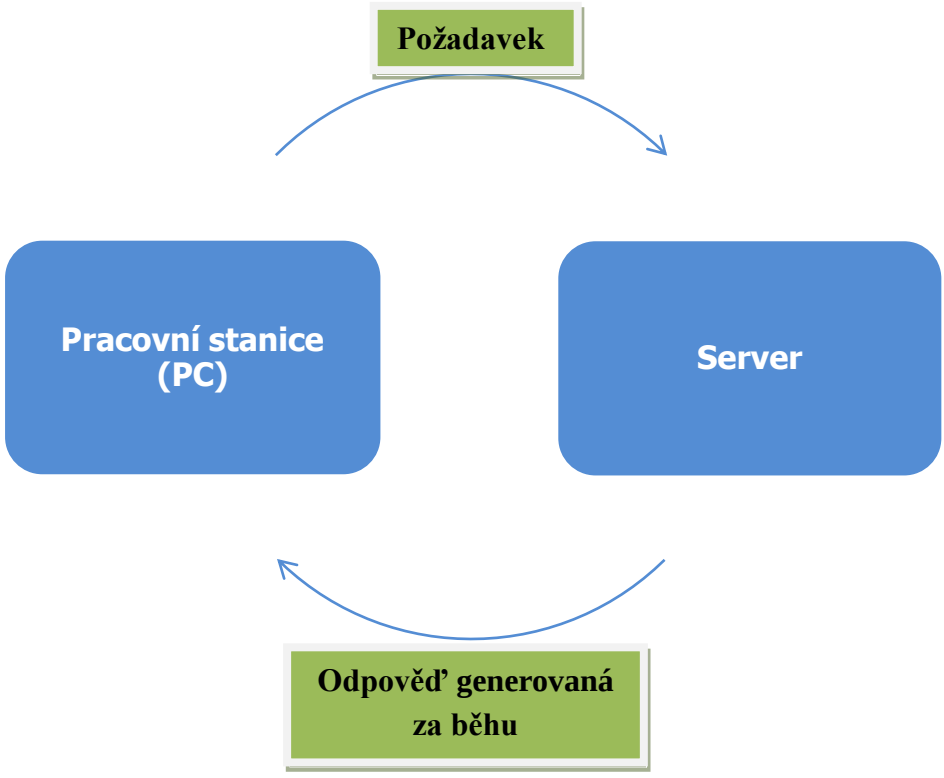
- **`Architektura „Multitier“`** – *je implementována podpora na rozdělení aplikaci na několik vrstev (tzv. „tiers“ – a jelikož je obecně těchto vrstev n, je často tato technologie nazývána **n-tier**). Tato technologie je postavena na základní třech vrstvách:*
  - ***Prezentační vrstva*** *– interakce s uživatelem – zobrazování dat a získávání dat od uživatele (+validace vstupních dat).*
  - ***Střední vrstva*** *– zde se provádí logické operace (např. u účetní aplikace výpočet výplaty z odpracovaných hodin).*
  - ***Vrstva pro přístup k datům*** *– všechny aplikace potřebují někde ukládat a číst data a k tomu je určena tato vrstva.*
- **`Nezávislost na výrobci`** *– tato verze (jako i všechny ostatní verze Javy) není uzamčena pouze pro jednu platformu.*
- **`Škálovatelnost`** *– nabízí flexibilní systém, který je možný upravovat dle potřeby (zátěže, kapacity, ...)*

5.4.2 Funkce

Základním pilířem je koncept klient – server. Klientem může být konzolová aplikace napsaná v Javě, grafický program (ty jsou nazývány „tlustými“ klienty) nebo webová aplikace („tenký“ klient). Server může být ve formě webových a výpočetních komponent. Webové komponenty jsou ve formě JSP nebo servletů.

5.4.3 Java Servlety

Servlet je technologie k obohacení serverů. Pokud si představíme jednoduchý případ statického serveru. Například pomocí webového prohlížeče pošlete požadavek na server, který vrátí odpověď. V tomto případě se neuvažuje možná změna informace – takže se nejedná o dynamický systém, který ale v převážné většině je pro nás výhodnější. Servlety jsou implementovány pomocí třídy ***`javax.servlet.Servlet.`***



Obr. 5: Dynamický model

V této práci je servlet využit pro interpretaci – jelikož v konfiguraci CLDC není implementována třída pro práci s SQL. Servlet přijme data pomocí http spojení a tyto data převede na SQL příkazy a výsledky (potvrzení o vykonané práci příp. o problému) převede zpět a odešle na zařízení. Druhý servlet je na výpis obsahu databáze. V daném případě se jedná o optimální řešení, jelikož implementace SQL by zabrala další místo v paměti a také výkon zařízení. Zdrojový kód lze nalézt v *Program/Java EE*.

5.4.4 JavaServe Pages (JSP)

JSP je textový dokument založený na HTML jazyce s obsahem Java kódu. Obecně se dá říci, že JSP je „skriptovací jazyk“ podobný PHP. Jedná se v podstatě o nadstavbu servletů – dost často tyto dva systémy spolu spolupracují – na pozadí běží servlet a o zobrazování informace se stará JSP. JSP je v mé prácí využito přesně takto. K databázi jde přistoupit i pomocí internetu. Přístup je z několika důvodů:

- pacient nemusí jít k lékaři, aby si on stáhnul naměřené hodnoty
- úspora paměti
- záloha v případě neštěstí
- lékař (příp. někdo další) může pacienta sledovat „na dálku“

Přístup je samozřejmě umožněn pouze pacientovi a on sám může rozhodnout, komu dalšímu povolí přístup. Jak již bylo zmíněno, na pozadí běží servlet, který je v roli interpreta pro práci s databází a „vzhled“ stránek je generován pomocí JSP a také zabezpečení je pomocí JSP.

```
String jmeno=rs.getString(1);
String heslo=rs.getString(2);

if(uzivatel.equals(jmeno) && identifikace.equals(heslo))
{                                   %>
       <jsp:forward page="databaze" />
   <%}
else
   out.println("Prihlaseni se nepovedlo. Zkuste to znovu");
```

Tato část kódu je z JSP, který slouží pro zabezpečeného přihlášení. Dochází k načtení zadaných údajů, porovnání a v případě shody dochází k přesměrování na servlet, který vypisuje obsah databáze.

6. Čidla

V této kapitole budou rozebrána použitá čidla. Informace převážně z katalogů jednotlivých součástek. Použité externí zdroje [19] (6.1 Akcelometr), [20] (6.2 Čidlo tepu a saturace kyslíkem a [21] (6.3 Teploměr).

6.1 Akcelometr

Akcelerometr je senzor pro měření zrychlení:

- **Dynamické zrychlení** – *resp. sílu vzniklou změnou rychlosti pohybujícího se předmětu.*
- **Statické zrychlení** – *resp. sílu vzniklou působením zemské gravitace.*

Typů akcelerometrů je několik – podle počtu os (1D, 2D a 3D), podle citlivosti, podle principu funkce (piezoelektrické, piezorezistivní, tepelné a MEMS[7]) , … Oblast použití akcelerometrů je široká – například:

- **Automobilový průmysl** *(airbagy, ESP)*
- **Měření vibrací** *(ložiska, notebooky)*
- **Měření natočení**
- **Měření rychlosti**
- **Navigační systémy**

6.1.1 Typy

Základní rozdělením akcelerometrů je podle počtu os:

- **1D** – *měření v jedné ose. V dnešní době by se mohlo zdát, že jde o zbytečné zařízení, ale stále má své využití kvůli vyšší hodnotě zrychlení, odolnosti a také ceně.*
- **2D** – *v principu pracuje jako dva 1D senzory otočené o 90°. Je vhodný pro měření vibrací, pohybu po podložce.*
- **3D** – *měření ve třech osách (x, y a z). Obecně se vyrábí pro menší hodnoty zrychlení - proto v krajních případech musíme použít kombinaci jednoho 2D a jednoho 1D senzoru.*

[7] Micro Electro Mechanical Systems

6.1.2 Použitý akcelometr MMA7260QT

Použitý akcelometr je od firmy Freescale Semiconductors. Jedná se o kapacitní mikromechanický akcelometr s možností nastavení citlivostí a tepelnou kompenzací. Informace čerpány z *Datasheet MMA7260QT* [22].

Vlastnosti

- **Nastavitelná citlivost:** 1.5/2/4/6g
- **Malý proudový odběr**: 500 µA (Sleep Mode: 3 µA)
- **Nízké napájecí napětí:** 2.2 – 3.6 V
- **QFN pouzdro:** 6mm x 6 mm x 1.45mm
- **Vysoká citlivost:** 800mV/g @1.5 g
- **Teplotní rozsah:** -40 až 105 ºC

Obr. 6:QFN pouzdro

Popis

Na obrázku lze vidět zapojení. Piny 1 a 2 jsou tzv. *g-Select1 (resp.2)*. Pomocí těchto pinů lze nastavit citlivost přístroje v závislosti na logickém vstupu na tomto pinu. Je to výhodná vlastnost pro systémy, kde potřebujeme pracovat s proměnnou citlivostí. Zde jednoduchou změnou vstupu (změnou úrovně vstupu) dojde ke změně citlivosti. Nastavení dle tabulky:

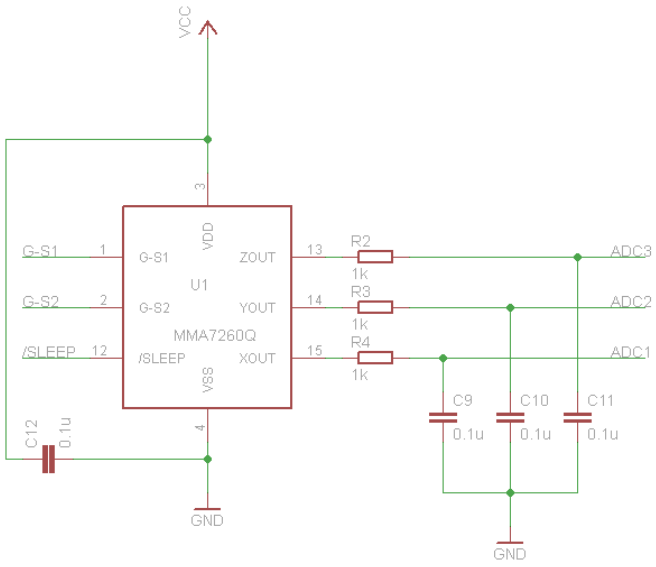
g-Select1	g-Select2	Rozsah	Citlivost
0	0	1.5g	800mV/g
0	1	2g	600mV/g
1	0	4g	300mV/g
1	1	6g	200mV/g

Tabulka 7: Nastavení citlivosti

Pin 4. je vstup napájení, pin5. je zem. Piny číslo 5-11,16 bez využití, pin 12 je pro uspání, pin 13 je napěťový výstup osy *z*, pin 14 napěťový výstup osy *y a* pin 15 napěťový výstup osy *x*.

Zapojení

Akcelerometr je propojen s mikrokontrolér ATmega 8L. Výstupy jednotlivých os jsou přivedeny na vstupy A/D převodníku mikrokontroléru. I přesto, že je používán pouze jeden výstup, jsou přivedeny všechny – pro možné další rozšíření činnosti (možnost kontroly, zda pacient udržuje dostatečný pohyb, …). Na výstupu jednotlivý os jsou RC filtry – pro omezení šumu od hodin.



Obr. 7: Zapojení akcelerometru

Cílem akcelometru je zachytit pád člověka, proto nám stačí pouze osa z (za předpokladu stále stejné orientace zařízení – to můžeme předpokládat, jelikož člověk toto zařízení bude mít u pasu, přichycené nějakým opaskem). A poté je důležité odlišit stav, kdy si člověk pouze sedne a kdy se již jedná o pád. Stoprocentně to odlišit nejde, proto je použita metoda „více falešných poplachů“, kdy je mez nastavena níže s tím, že to někdy zareaguje i na falešný impulz, ale zase máme jistotu, že to zachytí případný pád. Z tohoto důvodu je také připojeno tlačítko pro reset zařízení a to hned z dvou důvodů:

- **`Rychlé sednutí`**
- **`Pád`** – *pacient je v pořádku, není potřeba volat záchrannou službu.*

Hodnota je průměrována z 50-ti měření – tím se odstraní chvilkové překmity. Neovlivní se tím funkce – pád člověka (i nekontrolovaný pád, kdy je člověk v bezvědomí) bezpečně přesáhne toto období.

Je implementován filtr pro odhalení falešných impulzů – např. pokud člověk se rozeběhne, bude poskakovat, atd. Implementace je vidět v kapitole 8.2 Software pro řízený blok.

6.2 Čidlo tepu a saturace kyslíkem

Je to neinvazivní (nedochází k průniku kůží) metoda. Senzor se umístí na úzkou část na těle – prst nebo ušní lalůček. Je vysíláno záření o dvou vlnových délkách (první v pásmu 600 – 750 nm, druhé v infračervené oblasti 850 – 1000 nm) a na druhé straně je přijímáno fotodetektorem. Změna absorpce každé vlnové délky je měřena – z toho údaje je vyhodnocen pulzace arteriální krve. Pokud je hladina kyslíku v krvi nízká, absorbuje více světla (v nižším pásmu), ale prochází více infračerveného záření. Po průchodu signálu je spočten poměr záření a výsledek je poté vybrán z implementované tabulky (většinou si dělá každý výrobce sám na základě měření). Typické hodnoty jsou od 95 do 99% (pro lidi s chronickou obstrukční plicní nemocí jsou hodnoty od 88 do 95%). Přídavné absorpce jako kůže, kosti, žíly jsou stále přítomny a jejich hodnota se nemění, proto se dají odečíst. V případě tepu srdce se na okamžik zvýší objem krve v místě měření a to se projeví ve změně absorpce. Jediný faktor ovlivňující (či zabraňující) je podchlazení – pokud je člověk podchlazen, nemusí být vůbec možné měření provést (z důvodu nedostatečného prokrvení periferií). Zařízení komunikuje po rozhraní UART a posílá data jednou za sekundu v rámci:

```
<100.0,98.0>
```

- `100.0` – *tep.*
- `98.0` – *saturace krve v procentech.*

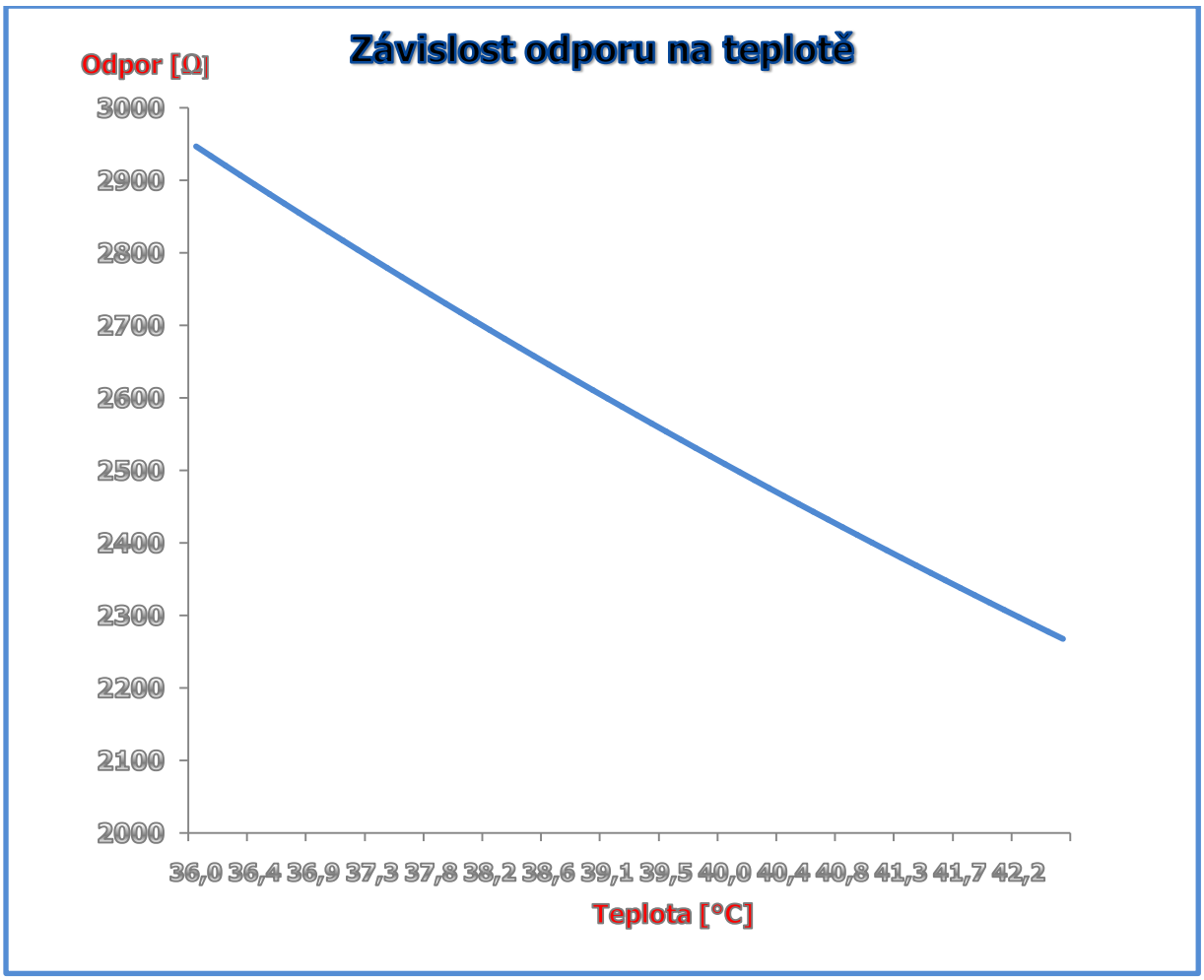
6.3 Teploměr

Jako teploměr byl vybrán termistor NTC640-4k7. Jak je již vidět z názvu jedná se o termistor s negativní teplotní závislostí, což znamená, že se stoupající teplotou, klesá odpor. Jedná se o termistor od společnosti Vishay BCcomponents. Informace čerpány z *Datasheet NTC640.pdf* [23].

6.3.1 Vlastnosti

Parametry:

- **Teplotní rozsah:** *-40°C až 125°C*
- **Odpor při 25°C:** *4700* Ω
- **Tolerance:** *±3%*
- **Rozptylový faktor:** *7mW/K*
- **Negativní závislost odporu na teplotě**



Graf 1: Závislost odporu na teplotě

Na grafu lze vidět závislost odporu na teplotě – data jsou po výpočtu Steinhart-Hart aproximace (veškeré koeficienty dostupné v [23]).

6.3.2 Steinhart-Hart aproximace

Pro přesné měření teploty byla v roce 1968 vyvinuta rovnice nazvaná podle svých objevitelů – Steinhart-Hart, která slouží k výpočtu poměru odpor-teplota. Rovnice jsou převzaty z [23].

$$R(T) = R_{ref} \cdot e^{(A+\frac{B}{T}+\frac{C}{T^2}+\frac{D}{T^3})}$$

$$T(R) = A_1 + B_1 \cdot \ln\frac{R}{R_{ref}} + C_1 \cdot \ln(\frac{R}{R_{ref}})^2 + D_1 \cdot \ln(\frac{R}{R_{ref}})^3$$

Kde:

A,B,C,D,A_1, B_1, C_1, D_1 .....jsou konstanty závislé na typu materiálu

R_{ref} … odpor při referenční teplotě 25°C *(4700 Ω)*

T … teplota v Kelvinech

6.3.3 Zapojení

Zapojení lze vidět na schématu v příloze. Je použit jednoduchý princip odporového děliče. Na odporu je snímáno napětí a přivedeno na A/D převodník. V programu je použita Steinhart-Hartova aproximace. A/D převodníkem je snímáno napětí, to je převedeno na odpor (pomocí vztahu pro odporový dělič) a z něj je spočítána teplota. Následovně převedena na stupně celsia. Teploměr je kalibrován pomocí lékařského teploměru – je kalibrován ve třech bodech.

7. Praktická realizace

Pro účely tohoto zařízení byly navrženy a vytvořeny dvě desky plošných spojů. Budou zde probrány dílčí části. Celé schéma lze nalézt v *DPS/DiplomovaPrace_Cinert.sch*.

7.1 Mikrokontrolér

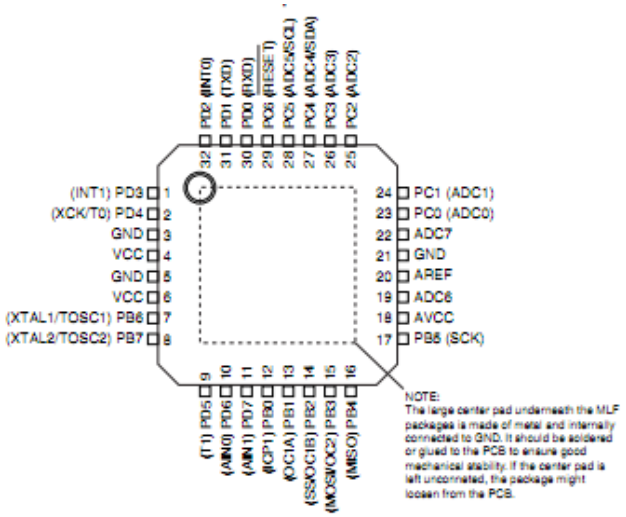
Řídící jednotkou je mikrokontrolér ATmega8L. Jedná se o CMOS 8-bitový nízko příkonový mikrokontrolér založený na AVR RISC architektuře. Tento mikrokontrolér pomocí spouštění výkonných instrukcí v jednom hodinovém cyklu dosahuje 1MIPS/MHZ. Informace v této kapitole čerpány z *Datasheet ATmega8* [24].

7.1.1 Vlastnosti

Klíčové vlastnosti:

- **Nízko příkonový 8bitový mikrokontrolér**
- **Vylepšená RISC architekura:**
  - *130 výkonných instrukcí*
  - *32x8 registrů*
  - *Až 16MIPS při 16MHz*
- **8 KB In-System Self-Programmable flash paměti**
- **512 B EEPROM**
- **1 KB vnitřní SRAM**
- **Zápisové/Mazací cykly**
  - *10 000 u paměti Flash*
  - *100 000 u paměti EEPROM*
- **Uchování dat:**
  - *20 let při 85ºC*
  - *100 let při 25ºC*
- **Periférie:**
  - *Dva 8bitové časovače/čítače*
  - *Jeden 16bitový časovač/čítač*
  - *3 PWM kanály*
  - *8mi kanálový ADC s 10bitovým rozsahem*
  - *Programovatelný USART*
  - *Master/Slave SPI*

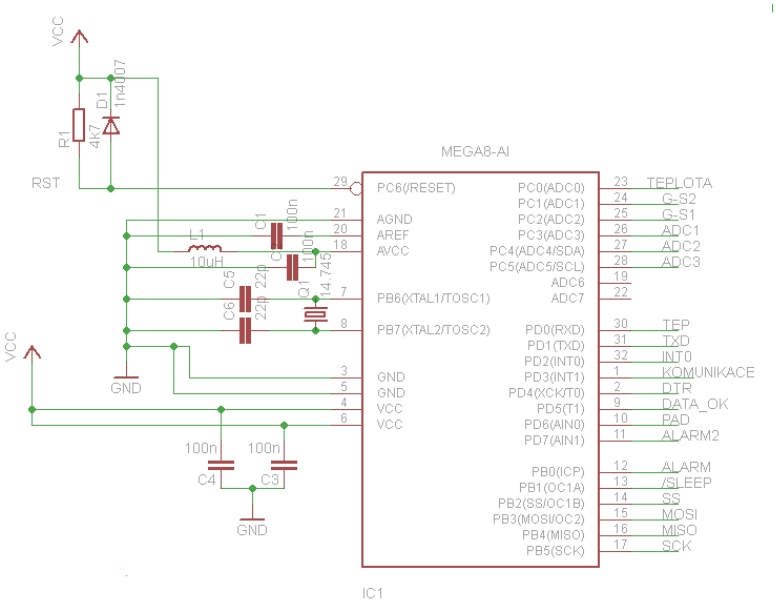
- **Napájecí napětí:** od 2.7 do 5.5V



Obr. 7: Konfigurace pinů [24]

7.1.2 Použití

V levé části můžeme vidět zapojení napájecích obvodů, krystalu a resetu. Jedná se o doporučené zapojení dle dokumentu *AVR Hardware Design Considerations* [25]. Na pravé straně jsou jednotlivé piny. Začneme podle jednotlivých portů. V závorce pojmenování dle schématu.



Obr. 8: Zapojení ATmega8

PORTB

PB0 ***(ALARM)*** - použit pro signalizaci alarmu pomocí LED diody ALARM.

PB1 ***(/SLEEP)*** - pro uspání akcelerometru .

PB2 ***(SS)*** - pro výběr podřízeného zařízení (SS -*Slave Select*), pro komunikaci SPI.

PB3 ***(MOSI)*** - pro programování ISP a pro komunikaci po SPI. Je to pin MOSI – *Master Output Slave Input.*

PB4 ***(MISO)*** - stejné použití jako předcházející. MISO – *Master Input Slave Output.*

PB5 ***(SCK)*** - pro programování ISP.

PORTC

PC0 ***(TEPLOTA)*** – první vstupní kanál A/D převodníku. Snímaná hodnota napětí je převedena na teplotu.

PC1 ***(G-S2)*** – použit jako nastavení citlivosti akcelerometru.

PC2 ***(G-S1)*** – stejné použití jako u předcházejícího.

PC3 ***(ADC1)*** – třetí vstupní kanál A/D převodníku. Použit pro snímání napětí z akcelerometru – osa x.

PC4 ***(ADC2)*** - čtvrtý vstupní kanál A/D převodníku. Použit pro snímání napětí z akcelerometru – osa y.

PC5 ***(ADC3)*** - pátý vstupní kanál A/D převodníku. Použit pro snímání napětí z akcelerometru – osa z.

PORTD

PD0 ***(TEP)*** – čtení dat z UART (čidlo tepu a saturace kyslíkem).

PD1 ***(TXD)*** – posílání dat po UART. Odesílání dat do řídícího bloku.

PD2 ***(INT0)*** – přerušení 0. Připojeno na tlačítko pro zrušení alarmu.

PD3 ***(KOMUNIKACE)*** – přerušení 1. Pro změnu komunikace při selhání primárního komunikačního kanálu.

PD4 ***(DTR)*** – signalizace dostupných dat. DTR (*Data Terminal Ready*). V neaktivním stavu v nízké úrovni.

PD5 ***(DATA_OK)*** – signalizace správně přijatých dat. Alternativní funkce je signalizace dostupných dat při SPI komunikaci (jelikož uvažujeme nemožnost použití DTR).

PD6 ***(PAD)*** – signalizace alarmu (pád). V neaktivním stavu ve vysoké úrovní – změna do nízké = alarm.

7.2 Pomocné obvody

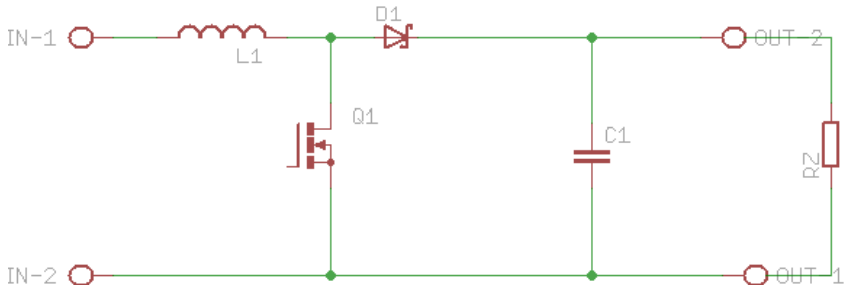
Pod pojmem pomocné obvody jsou skryty obvody pro úpravu napěťových úrovní (TTL=>RS232), pro úpravu napájecího napětí (na 3.3V) a signalizační obvody. Další částí (konektory, pomocné odpory) zde nebudou popisovány.

Převodník úrovní je postaven na obvodu MAX232 (Datasheet MAX232.pdf [26]) doplněný kondenzátory pro funkci nábojové pumpy. Stabilizátor napětí je postaven na obvodu LF33CV (Datasheet LF33CV [27]) – stabilizuje napětí na 3,3V.

7.2.1 Napájení

K napájení GSM modulu je použit DC/DC měnič – konkrétně se jedná o zvyšující měnič tzv. step-up měnič. Informace v této kapitole čerpány z [28] a *Datasheet LM2577* [29].

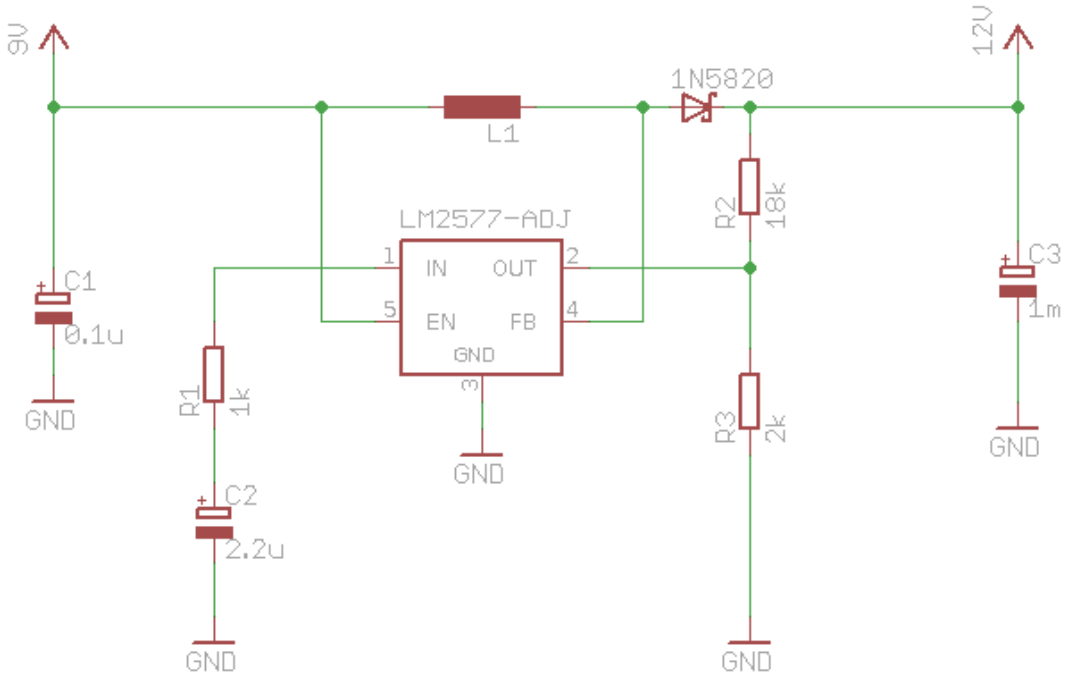
Principiální schéma lze vidět na následujícím obrázku.



Obr. 9: Zvyšující měnič

Během sepnutí spínače (tranzistoru Q1) roste proud cívkou L a zvyšuje se energie akumulovaná v energetickém poli cívky – cívka se chová jako spotřebič. V druhé fázi – rozepnutí spínače – protéká proud ze zdroje IN a také z cívky do zátěže R_z. Cívka se teď chová jako zdroj. Dioda zabraňuje vybíjení kondenzátoru přes sepnutý transistor. Výstupní napětí je vyšší než vstupní.

Toto zapojení bylo realizováno pomocí obvodu LM2577, který má vnitřní oscilátor naladěn na 52 kHz. Schéma je vidět na následujícím obrázku:



Obr. 10: Realizace zvyšujícího měniče

Jednotlivé komponenty byly vypočítány pomocí vzorců dostupných v [29]. Odpor R1 a kondenzátor C2 jsou z důvodu stability obvodu. Jedná se obvod LM2577-ADJ, kde ADJ znamená nastavitelný obvod. Výstupní napětí je nastaveno výstupním děličem podle vzorce:

$$\frac{R_2}{R_3 + R_4} = \frac{U_{výst}}{1.23} - 1$$

Na vstup je připojena 9V baterie a na výstupu je 12 V, což je doporučené napájecí napětí pro GSM modul. Obvod je spočítán na maximální proud 600 mA, kdy GSM modul má maximální odběr 500 mA při odesílání dat GPRS 4mi kanály.

Za vstupem je obvod, který signalizuje vybití baterie pod určitou hodnotu. Jedná se o napěťovou referenci – při poklesu LED dioda přestane svítit.

8. Software

Software je složen z hlavních dvou částí – ze dvou bloků:

- **Řídící blok** – *napsán v Javě, běží v GSM modulu a pomocí něj ovládá měřící desku.*
- **Řízený blok** – *v jazyce C, v mikrokontroléru ATmega. Obsluhuje měření, vyhodnocuje prvotní podněty pro alarm a příp. přeposílá tyto informace k dalšímu posouzení do řídícího bloku.*

Tato celá kapitola ukazuje ukázky kódu – většinou se jedná o zkrácenou verzi, která má za cíl ukázat hlavní princip dané metody.

8.1 Software pro řídící blok

Program je složen z 8 tříd:

- **Databaze** – *vytvoření a správa databáze.*
- **Inicializace** – *inicializace GSM modemu a měřící desky.*
- **Jadro** – *hlavní program obsluhující ostatní třídy.*
- **Pripojeni** – *vytvoření GPRS spojení a odeslání dat do vzdálené databáze.*
- **SPI** – *vytvoření SPI spojení a načtení dat.*
- **Spojeni** – *načtení a ověření dat.*
- **Start** – *spouštěcí metoda.*
- **WatchDog** – *kontroluje program a také funkčnost programu.*

A dvou knihoven:

- **Stav** – *„rozhraní“ mezi vlákny. Metody pro nastavení sdílených proměnných jsou synchronizovány – tím je docíleno toho, že s danou proměnnou lze pracovat pouze z jednoho vlákna.*
- **Zaznam** – *pro záznam průběhu programu. Ukládá data do textového souboru. Ponechává jeden starý záznam. Ukázku lze vidět na následující stránce.*

Program uchovává dva záznamy – současný (`log.txt`) a předcházející (`log.stary.txt`). Ukázku několika záznamů lze vidět v příloze. V první fázi dochází k smazání starého logu (pokud existuje), poté k vytvoření nového logu a příp. přejmenování předcházejícího logu. Otevírání záznamu je pomocí `FileConnection`, jak již z názvu vypovídá, slouží k tvorbě spojení se soubory. Je volána metoda `Connector.open()`, která má dva parametry – cestu k souboru a typ přístupu (čtení nebo čtení i zápis).

```
public class Zaznam {
	FileConnection log;
	PrintStream zapisLog;

	public Zaznam() {

		try {
			smazaniStarehoLogu();
			log = otevritLog("a:/log.txt");
			if (log.exists()) {
				log.rename("log.stary.txt");
				log = otevritLog("a:/log.txt");
			}
			log.create();
			zapisLog = new PrintStream(log.openOutputStream());
		} catch (IOException ex) {
			problem = true;
		}

	}

	private FileConnection otevritLog(String jmenoSouboru) throws
                                                    IOException {
		if (log != null) {
			log.close();
		}
		 return ((FileConnection) Connector.open("file:///" +
                                jmenoSouboru,Connector.READ_WRITE)
                                );
	}
	public synchronized void zapis(String zaznam, typZapisu typ) {

		if (typ.equals(typZapisu.Informativni)) {
		     zaznam = typZapisu.Informativni.name +
                        aktualniCas()+"  "+ zaznam;
		} else if (typ.equals(typZapisu.Chybovy)) {
		//...
		}
		zapisLog.println(zaznam);
//...
	}
```

Poté je již proveden zápis pomocí metody `zapis(String Zaznam,typZapisu typ)`. První parametr je řetězec určený k zapsání, druhý je typ zápisu:

- **`Informativní`** – *záznam průchodu programu jednotlivými sekcemi.*
- **`Chybový`** – *převážně výpis generovaných výjimek.*
- **`Alarm`** – *výpis při generaci alarmu.*

Součástí každého zápisu je také aktuální čas. Část ukázkového logu lze vidět na následujícím obrázku:

```
[II]Tue Apr 01 12:07:17 UTC 2012  Seriove spojeni otevreno.

[II]Tue Apr 01 12:07:18 UTC 2012  Prijata data v poradku.

[II]Tue Apr 01 12:07:19 UTC 2012  Data:121.8,98.1,36.7,N


[EE]Tue Apr 01 12:07:35 UTC 2012  Upload se neprovedl z nasledujiciho
duvodu:
java.io.IOException: Profile could not be activated
```

Záznam začíná typem zápisu, poté následuje čas zápisu a samotná zpráva. V první části lze vidět informativní zápisy z čtení dat ze sériového portu. V druhé části je vidět chybový výpis při problému s GPRS spojením (nemožnost nastavit GPRS profil).

8.1.1 Třída Start

Jak již bylo popsáno v sekci o Javě ME, spouštěcí třídou jsou MIDlety a místo metody *main* je zde metoda *startApp*. Po zapnutí zařízení se spouští ***Start.jad*** v GSM modulu s nastaveným zpožděním 10 sekund – z důvodu nalezení a zaregistrování do sítě GSM.

V metodě startApp() jsou vytvořena a spuštěna dvě vlákna:

- **Jádro** – *hlavní program starající se o vše ohledně chodu.*
- **Watchdog** – *hlídá běžící program a při problémech ukončí program (periodicky testuje program).*

```
public void startApp() {
        stav = new Stav();
        zaznam = new Zaznam();
        /* Vytvoreni vlaken */
        watchdog = new WatchDog(stav, zaznam);
        jadro = new Jadro(stav, zaznam);
        /* Start vlaken */
        watchdog.start();
        jadro.start();
```

Obě třídy implementují rozhraní `Runnable`, které umožňuju „běhuschopnost“ těchto tříd. Zjednodušeně to znamená, že je možné je spustit jako samostatné vlákno. Watchdog by měl běžet ve vlastním vlákně – pokud by bylo vše v jednom vlákně a to by se zablokovalo, tak by byl watchdog bezmocný. Dále nám to umožňuje jednoduchou správu běhu programu a ovládání jiných vláken (příp. tříd). V neposlední řadě v metodě `startApp()` by neměl být žádný kód, který by mohl způsobit problém (odebrání všech dostupných prostředků) a proto volání jiného vlákna je ideálním řešením. Oba konstruktory tříd mají shodné parametry – knihovny `Stav` a `Zaznam`.

Druhou důležitou částí třídy ***Start*** je metoda `destroyApp()`, která slouží k ukončení programu. Tato metoda je volána po stisku tlačítka vypnutí.

```
public void destroyApp(boolean unconditional) {
        zaznam.zapis("Konec stisknutim tlacitka", typZapisu.Informativni);
        watchdog.stop();
        jadro.stop();
        zaznam.uzavreni();
        /* Uvolneni prostredku */
        notifyDestroyed();
    }
```

Podle informací od výrobce je po stisku tohoto tlačítka čas zhruba 5 sekund na ukončení činnosti a poté dojde k vypnutí modemu. Tato metoda obsahuje informativní výpis o zmáčknutí tlačítka a poté dvě metody k zastavení vytvořených vláken. Jednotlivé činnosti jako např. zavření spojení, uzavření databáze atd. mají na starosti jednotlivá vlákna. Vlákno není ukončeno okamžitě – ukázka z třídy ***Jadro***:

```
    public void stop() {
            ukonceniProgramu();
            try {
                vlaknoHlavni.join(cas);
            } catch (InterruptedException ex) {
                vlaknoHlavni.yield();
            }
        }
```

Je použita metoda `join(cas)`, která čeká předepsaný čas (parametr `cas`) a poté pošle vláknu signál **interrupt**. Vlákno je poté „donuceno“ uvolnit prostředky a skončit pomocí metody `yield()`. V tomto případě je nastaveno na čekání 1 sekundu. Na závěr

dochází k uzavření záznamu (textového souboru) a poté k uvolnění všech prostředků přidělených tomuto MIDletu pomocí metody `notifyDestroyed()`.

8.1.2. Třída Jadro

Ve třídě ***Jádro*** je deklarace proměnných a funkční kód. Dochází k dílčím inicializacím a příp. řešení vzniklých problémů. Celý běh programu je zaznamenáván do záznamu. Tato třída slouží jako obslužná třída – spouští, ovládá a koriguje ostatní třídy a řeší vzniklé situace. Celý koncept je postaven na generování výjimek a většina těchto výjimek je předána této třídě a ta vzniklou situaci řeší. Je to výhodné, protože dochází k řešení problému na jednom místě, lze využít i jiné třídy. V případě neřešitelného problému (z hlediska softwaru) lze využít jiné třídy k realizaci záložního plánu (např. při nefunkčnosti hlavního komunikačního prostředku lze využít záložní komunikace přes SPI pomocí třídy ***SPI***).

```
public Jadro(Stav reset,Zaznam zaznam) {
        buffer=new StringBuffer();
        stopky=new Timer();
        /* Program startuje */
        zaznam.zapis("Hlavni program startuje",typZapisu.Informativni);
        zaznam.zapis("Inicializace modemu",typZapisu.Informativni);
        /** Inicializace modemu */
        try {
            init=new Inicializace(reset,zaznam);
            zaznam.zapis("Inicializace modemu
                          OK",typZapisu.Informativni);
             init.ATsend("ATD+xxxxxxxxx\r");
        } catch (IllegalStateException ex) {
            zaznam.zapis("Inicializace
                          modemu"+ex.toString(),typZapisu.Chybovy);
            reset.setStav(true);
        }
        init.at.addListener(this);

        /* Seriove spojeni */
        zkouskaSpojeni();

        /* Inicializace Databaze */

         zaznam.zapis("Inicializace databaze",typZapisu.Informativni);

        try {
            databaze=new Databaze();
            zaznam.zapis("Databaze obsahuje" +databaze.pocetZaznamu() +
                         "zaznamu",typZapisu.Informativni);
      } catch (RecordStoreException ex) {
```

Jedná se pouze o ukázku části třídy – více v jednotlivých třídách a jejich použitích. Většina kódu je v bloku `try-catch`, který „zkouší“ spustit kód a pokud dojde

k nestandardnímu chování, kód generuje výjimku, která je řešena v bloku `catch`. To koresponduje s tím, co bylo popsáno výše – výjimky jsou delegovány výše až do třídy ***Jadro***. V první části dochází k inicializaci modemu a zde při vygenerování výjimky nelze mnoho dělat, jelikož se převážně jedná o hardwarový problém – jediné řešení zkusit restart a informovat technika. Dále dochází k inicializaci sériového spojení a otestování příjmu dat pomocí metody `zkouskaSpojeni()`.

```
private void zkouskaSpojeni(){
        vlaknoCteni(false);
        if(!reset.getData()) {
            zaznam.zapis("Nemame data. Spoustime cteni
                          znova",typZapisu.Chybovy);
            vlaknoCteni(false);
            if(!reset.getData()) {
                zaznam.zapis("Podruhe nebyly prijata data. Prepiname na
                              komunikace   po SPI",typZapisu.Chybovy);
                chybaKomunikace();
            }

        } else {
            try {
                init.setPin(3);
                init.setPin(2);
            } catch (IOException ex) {
//...

private synchronized void serioveSpojeni() {
        try {
           spojeni = (CommConnectionControlLines)
            Connector.open("comm:COM0;baudrate=4800", Connector.READ_WRITE,
            true);
            comListener=new CommConnectionControlLinesListener(){
                public void DTRChanged(boolean SignalState) {
                    if(SignalState){
                        init.ATsend("AT+CFUN=1,0\r");
                        vlaknoCteni(true);
                    } else {
                        init.ATsend("AT+CFUN=9,0\r");
                    }
                }
            };
            spojeni.addListener(comListener);
        } catch (IOException ex) {
            zaznam.zapis("Nepodarilo se nam otevrit seriove
                          spojeni",typZapisu.Chybovy);
        }
```

Tato část programu má za cíl vyzkoušet sériové spojení. Po inicializace GSM modulu dojde k inicializaci mikrokontroléru ATmega a poslední část je odeslání testovacího rámce,

který se snaží GSM modul zachytit a tím ověřit funkčnost spojení. Data mají předem definovaný rámec:

<100.0,98.0,37.0,N>

- **<** – *začátek rámce.*
- **100.0** – *tep (je se může pohybovat rozsahu 0 až 250 – proto máme tři číslice před desetinnou čárkou a jedno za).*
- **98.0** – *saturace kyslíkem (v procentech).*
- **37.0** – *teplota.*
- **N** – *signalizace pád (N – ne, A – ano).*
- **>** – *konec rámce.*

Tato metoda vytvoří čtecí vlákno `vlaknoCteni(false)`. Parametr `prijemDat` nám určuje, zda jde o testování spojení či o příjem dat – v tom případě dojde k uložení přijatých dat do databáze a proměnná pro watchdog je změněna (viz. kapitola o třídě Watchdog).

Pomoci metody `serioveSpojeni()` je vytvořeno spojení s danými parametry (4800 Bd, pro čtení i zápis, definovaný časový limit), je nastaven listener na toto spojení – pro čtení dat po uspání (změna na pinu DTR nám vyvolá dočasné vzbuzení a spuštění tohoto kódu).

Pokud jsou data přijata, je to potvrzeno vysokou úrovní na pinu číslo 7. V opačném případě, jsou data odeslána znovu. Pokud dvakrát po sobě nejsou přijata data, je tato situace vyhodnocena jako selhání spojení a je přepnuto na záložní komunikaci SPI – na to slouží metoda `chybaKomunikace()`.

```
private void chybaKomunikace() {
        init.poslaniSMS("Selhalo spojeni s deskou",2);
        init.poslaniSMS("Zkontrolujte,zda se neuvolnil komunikacni kabel",1);
        zaznam.zapis("Pokus uzavrit seriove spojeni a otevreni SPI
                    spojeni",typZapisu.Informativni);
        uzavreniSpojeni();
        otevreniSPI();
    }

private void otevreniSPI() {
        init.otevritSPI();
        try {
            init.zmenaPinu(true);
        } catch (IOException ex) {
```

Je odeslána SMS technikovi a uživateli. Poté je otevřena náhradní komunikace SPI pomocí metody `otevreniSPI()`. Tato metoda volá jednoduchý AT příkaz (viz. popis třídy Inicializace).

Po průvodní inicializaci je spuštěna hlavní smyčka:

```
public void run() {
        zaznam.zapis("Hlavni program nabiha", typZapisu.Informativni);
        init.ATsend("AT^SBV\r");
        zaznam.zapis("Napeti baterie je: "+init.getNapeti()+"
                    mV",typZapisu.Informativni);
        zaznam.zapis("Uspavame zarizeni. Cekame na
                    data",typZapisu.Informativni);
        stopky.scheduleAtFixedRate(new casVlakno(),start,perioda);
        init.signalizace(deska);
        init.ATsend("AT+CFUN=9,0\r");
}
```

Nejdříve je otestováno a vypsáno napětí baterie (při poklesu pod určitou mez je uživatel upozorněn signalizací LED diody a SMS zprávou). Poté je nastaveno vlákno pro odeslání dat – je nastaveno na spuštění každou hodinu. Pokud všechny dílčí inicializace proběhly v pořádku, je to signalizováno zelenou LED diodou na měřící desce – pomocí metody `init.signalizace(deska)`. A na závěr je modem uspán.

Na další ukázce je vnořená třída ***casVlakno***:

```
private class casVlakno extends TimerTask{
        public void run() {
            zaznam.zapis("Spoustime naplanovany
                        upload.",typZapisu.Informativni);
           vlaknoUpload(true);
           if(SPI)
           {
               SPI=false;
               init.reset();
               zkouskaSpojeni();

           }
           zaznam.zapis("Napeti baterie je :"+init.getNapeti()+"
                        mV",typZapisu.Informativni);
           init.ATsend("AT+CFUN=9,0\r");
        }
```

Jedná se o vnořenou třídu rozšiřující `TimerTask` – umožňuje časování vláken. V rámci této třídy je spuštěno vlákno pro odeslání dat. Pokud je zvolena záložní komunikace, jednou za čas dojde k ověření, zda primární komunikace není dostupná. Poté dojde opět k uspání.

Tato třída implementuje rozhraní `ATCommandListener`. Je to rozhraní k zachytávání URC pomocí metody `ATEvent(String Event)`

```
public void ATEvent(String Event) {
        if(Event.equalsIgnoreCase("^SPI:")){
            init.ATsend("AT+CFUN=1,0\r");
            vlaknoSPI(Event);
        }
    }

    public void RINGChanged(boolean SignalState) {
    }
```

Dostupnost dat je signalizována změnou úrovně pinu, která je vyhodnocena (viz dále). Poté je spuštěno SPI čtení a data jsou vráceny ve formě URC. A to je zachyceno touto metodou, která spouští speciální vlákno na čtení dat (viz popis třídy SPI).

8.1.3 Třída Inicializace

Tato třída slouží pro inicializaci modemu a měřící desky, poskytuje rozhraní pro využití AT příkazů a obstarává obsluhu odpovědí na AT příkazy. Součástí je i nastavení GPRS profilu a obsluha SPI komunikace (obojí přes AT příkazy).

```
    public Inicializace(Stav prijem, Zaznam a) throws IllegalStateException,
                                                      ATCommandFailedException,
                                                      IOException {

            at = new ATCommand(false);
            listener = new ATCommandResponseListener(){
                public void ATResponse(String Response){

                    } else if(Response.startsWith("^SIND:",2)){
                       aktualniCas=Response;
                    } else if(Response.startsWith("^SBV:",2)){
                        napetiBaterie=Response;
                    }
                }
            };
            ATsend("ATZ\r");
            ATsend("AT^SIND=nitz,1");
            ATsend("AT^SCFG=\"Userware/Watchdog\",1");
            inicializacePort();
```

V první fázi dochází k vytvoření instance `ATCommand at` s parametrem `false` (bez podpory CSD). Tato inicializace může generovat vyjímku `ATCommandFailedException` – tato výjimka je generována při selhání části zařízení (není možná softwarová náprava). Druhá výjimka je `IllegalStateException` a ta signalizuje, že příkaz byl generován v *nesprávný čas* (tzn. Java Virtual Machine nebyl ve stádiu běhu). Jak je vidět v ukázce ze třídy ***Jadro***, tak tyto výjimky jsou řešeny restartem zařízení. Bohužel jakákoliv signalizace technikovi není možná, jelikož SMS či volání lze realizovat pouze pomocí AT příkazů. Jediná informace uživateli je, pomocí LED diod. Třetí výjimka `IOException` je generována při problémech s otevřením GPIO sběrnice.

Dále je vytvořen *listener* (zachytává odpovědi na jednotlivé příkazy) – jedná se o anonymní vnořenou třídu. Kód je zkrácen, na příkladu je vidět reakce na odchycení odpovědi na příkaz k synchronizaci času s GSM sítí a dotaz na napětí baterie. Obecné doporučení pro obsluhu listeneru (jako obsluhy přerušení), je zdržet se v tomto kódu co nejkratší dobu. Nejlepší řešení je tedy si uložit získaná data příp. změnit nějaký příznak a řešení nechat na další části kódu. Dalším možným řešení je volání samostatného vlákna. V další části je vidět dotaz na uživatelský profil (pokud neexistuje, je vytvořen), dotaz na GSM hodiny, zapnutí watchdogu a spuštění metody `inicializacePort()`

```
public void inicializacePort() throws IOException {
        vstupniPin = new Vector(2);
        vstupniPin.addElement("GPIO4");
        vstup = new InPort(vstupniPin);
        vystupniPin = new Vector(4);
        uroven = new Vector(4);
        vystupniPin.addElement("GPIO5");
        uroven.addElement(Integer.valueOf("1"));
        // ...
        vystup = new OutPort(vystupniPin, uroven);
        vstup.addListener(vstupListener = new InPortListener()
            public void portValueChanged(int value) {
                if (value==2) {
                    ATsend("ATD+420604450956\r");
             // ...
                }
                else if(value==3)
                {
```

V první části dochází k inicializaci vstupního portu GPIO4 a poté výstupních GPIO 5, 6, 7 a 8. Na ukázce inicializace pouze jednoho výstupního pinu a je mu nastavená vysoká úroveň. Dále je přiřazen listener na změnu na vstupních pinech – ukázka je z listeneru při komunikaci po SPI (alternativní komunikace). Jak již bylo zmíněno, je zde použit pin GPIO6 jako vstupní pin pro oznamování dostupnosti dat. Možné stavy na následující tabulce.

GPIO4	GPIO6	Stav
0	0	Pád
0	1	Pád + Data
1	0	Provozní
1	1	Data

Tabulka 8: Vstupní stavy

Z tabulky lze vidět, že při stavu *01* na vstupu se nám sejdou oba stavy – ale to nám nevadí, jelikož stav pád je pro nás prioritní. Je vynechána část kódu, který znemožňuje generaci alarmu znova – byla by volána záchranná služba opětovně.

Další důležitou metodou je `ATsend()`, která obstarává odesílání AT příkazů a odchytává jednotlivé výjimky. Parametrem této metody je řetězec, který chceme odeslat. Při odesílání je nastaven listener, který bude případně reagovat na odpověď. Po odeslání je počkáno 100 ms (některé příkazy trvají déle).

```
public void ATsend(String ATprikaz) {
        try {
            at.send(ATprikaz, listener);
            Thread.sleep(cekani);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        } catch (ATCommandFailedException ex) {
            ex.printStackTrace();
        } catch (IllegalStateException ex) {
            ex.printStackTrace();
        } catch (IllegalArgumentException ex) {
            ex.printStackTrace();
```

Další důležité metody můžete vidět na dalším výpisu. Už se u nich zastavíme jen stručně. Metoda `setPin(int uroven)` je pro změnu úrovně pinu – pro generaci resetu, potvrzení přijatých dat, signalizaci problému. Metoda `nastaveniPripojeni()` nastaví GPRS profil a `otevritSPI()` generuje přerušení (GPIO 7 je připojen na INT1 mikrokontroléru) a je přepnuto na SPI komunikaci.

```
public void setPin(int uroven) throws IOException
    {
        try {
            vystup.setValue(uroven);
            Thread.sleep(ustaleni);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }

    public void nastaveniPripojeni()
    {
        ATsend("AT^SJNET=\"gprs\",\"wap.t-mobile.cz\",\"wap\",\"wap\",,0\r");
    }

    public void otevritSPI()
    {
        setPin(4);
        setPin(5);
        zmenaPinu(true);
        ATsend("AT^SSPI=1020,0000,0000,0000 <1000010>\r");
    }
```

8.1.4 Třída Spojeni

Tato třída slouží pro načtení dat ze sériového rozhraní a jejich ověření. Tato třída implementuje rozhraní *Runnable* – je spustitelná jako samostatné vlákno. Ve třídě ***Jadro*** je tato třída volána v metodě `vlaknoCteni(boolean prijemDat)`

```
private synchronized void vlaknoCteni(boolean prijemDat) {
        serioveSpojeni();
        if(cteciVlakno!=null){
            cteciVlakno=null;
        }
        cteciVlakno=new Thread(new Spojeni(zaznam, buffer,spojeni,reset));
        cteciVlakno.start();
        try {
            cteciVlakno.join(3000);
        } catch (InterruptedException ex) {
            cteciVlakno.yield();
        }
        ulozeniDat(prijemDat);
    }
```

V první části dojde k vytvoření sériového spojení s rychlostí 4800bd a je nastaven časový limit. Pomocí této metody je spuštěna třída ***Spojeni*** – parametry jsou `zaznam, buffer, spojeni` a `reset.` Toto spojení musí být vytvořeno v hlavním vlákně, protože obsahuje *listener*, který budí vlákno ze spánku, když jsou data připravena k přesunu. Obě dvě metody jsou synchronizované – sériové spojení musí být vytvořeno vždy jen jedno – v opačném případě by byla generována výjimka.

Na závěr je volána funkce `cteciVlakno.join(3000)` – vlákno je spuštěno a pokud do 3s nevrátí řízení zpět je mu poslán signál *interrupt* (přerušení činnosti). Je generována výjimka *InterruptedException* a je volána funkce `cteciVlakno.yield()` – tato metoda pozastaví současné vlákno a uvolní paměť a procesor pro jiné.

Metoda cteniDat

Tato metoda slouží k načtení a verifikaci dat. Ukázka na další stránce. Metoda je chráněna proti zablokování, protože čte data pouze, pokud jsou dostupná ve vstupním bufferu, pomocí metody `dostupnaData().` Tato metoda generuje výjimku `IOException`, pokud data nejsou dostupná (čtení je zahájeno až po příjmu příznaku dostupnosti dat). Pokud jsou dostupná data, čekáme na začátek rámce. Po přijetí stanoveného znaku načítáme vstupní data a ukládáme do bufferu. Pokud je první znak '!' jedná se o alarm – pokud je třikrát přijat alarm, je volána záchranná služba. Při načtení tohoto stavu, je spuštěna metoda `testDat()` – dochází k načtení dat, pokud jsou ve správném formátu, je inkrementován čítač alarmů a pokud máme tři alarmy po sobě, tak je volána záchranná služba.

V rámci cyklu `for` stále dochází k testování dostupnosti dat (může během přenosu dojít k přerušení/odpojení kabelu a došlo by k zablokování metody `read()`). Poté je testováno, zda byl přijat ukončovací znak (zda jsme přijali platná data). Pokud je vše v pořádku, je vrácen buffer, jinak je vráceno `null.` A GPIO6 je nastaven do vysoké úrovně (viz. dále). Implementace této metody do hlavní třídy byla již ukázána. Dále dochází k inicializaci databáze.

Po přijetí dat je ještě ověřeno, zda jsou data správná, tj. nejsou nulová – to znamená, že jedno (či více) čidel nefunguje. V tomto případě je inkrementován čítač chyb. Pokud jsou doručeny po sobě třikrát špatná data, je spuštěn obslužný program (restart desky, informování uživatele).

```
public synchronized String cteniDat() throws IOException
    {
      if(dostupnaData()){

                c = vstup.read();
                if(c=='<'){
                    zaznam.zapis("Mame start ramce",typZapisu.Aktualizace);

                    for(int i=0;i<18;i++){
                        if(dostupnaData()) {
                            c=vstup.read();
                            sb.append(c);

                        }else{

                             zaznam.zapis("Nedosly veskara data" ,
                                         typZapisu.Chybovy);
                            return null;
                        }
                    }

                    loop=false;
                    if(sb.charAt(17)!='>){
                        zaznam.zapis("Prijaty neplatna
                                     data",typZapisu.Chybovy);
                        return null;
                    } else {
                        sb.deleteCharAt(17);
                         a.zapis("Prijate data v
                        poradku.",typZapisu.Informativni);
                        a.zapis("Data:"+sb.toString(),typZapisu.Informativ
                        ni);
                        return sb.toString();
                    }

                }

                }else if(c=='!'){
                   return(testDat());
                }
```

8.1.5 Databáze

Tato třída slouží k vytvoření a správě databáze. Databáze je typu *RecordStore*, která je nezávislá na napájení a tudíž jsou data uchována i při vypnutí (vybití baterie). Je vytvořena resp. otevřena (toto určuje druhý parametr, v opačném případě by databázi nevytvořil) databáze se jménem `Databaze`, je nastaveno oprávnění pro zápis pouze pro tento MIDlet a je povolen zápis. Poté je vytvořen list - jedná se o *RecordEnumeration*. Udržuje údaje o

databázi – automaticky při zapsání/smazání prvku se obnovuje. Pokud dojde ke smazání prvku, jeho index již není znovu použit, takže při procházení databází je zde prázdné místo. Toto obchází RecordEnumeration, který udržuje záznamy konzistentní. S jeho pomocí jsou pak čteny jen záznamy s daty a ne prázdné místa.

```
Databaze(){
        try {
            this.databaze=
             RecordStore.openRecordStore("Databaze",true,RecordStore.AUTHMODE_
             PRIVATE,true);
            list = databaze.enumerateRecords(null, null, true);
        } catch (RecordStoreException ex) {
            ex.printStackTrace();
        }
```

První index v této databázi je vyhrazen k vnitřním potřebám. Data jsou nahrávána do vzdálené databáze jednou za hodinu a tudíž, pokud dojde k vypnutí zařízení v této době, tak vznikne nekonzistence mezi daty v lokální a vzdálené databázi. Proto je v první pozici uložen index naposledy odeslaných dat a v dalším kroku jsou data odesílána od následujícího indexu. Toto je zařízeno pomocí metody `odeslano()` .

Metoda odeslano

Metoda využívá metody `cteniDB()`, která načítá data pomocí datového streamu - ukázka použití :

```
for(i=index;i<databaze.getNumRecords();i++)
 {

            bin = new ByteArrayInputStream(databaze.getRecord(i));
            din = new DataInputStream(bin);
            temp[1][posl-i] = Double.toString(din.readDouble());
            //…
            din.close();
            bin.close();
`
```

Dojde k vytvoření „propojení“ s určeným záznamem v databázi (určeno pomocí proměnné `i` až do konce databáze). V příkladu je vidět načítání hodnoty double (hodnota double je převedena na String). Na závěr je testováno, zda všechna data byla odeslána (čekáme na potvrzení ze serveru a poté je zapsán poslední odeslaný index do prvku jedna v databázi).

Metoda `odeslano()` na začátku testuje proměnnou `posl`, která určuje poslední odeslaný prvek (příp. je nula, pokud je databáze prázdná nebo jde o první průběh touto metodou). Je vytvořeno spojení do databáze, a pokud v první prvku není nic, tak je načítáno od druhého prvku. Důvody můžou být dva:

- **Databáze je prázdná.**
- **Z databáze nebyly ještě odeslány žádné data.**

Pokud vše proběhne jak má, dojde k načtení neodeslaných dat a k odeslání.

```
public String[][] odeslano() throws Exception
    {
        int i;
        try {
            if(posl==0)
            {

                bin = new ByteArrayInputStream(databaze.getRecord(1));
                din = new DataInputStream(bin);
                i=din.readInt();
                if(i==0)
                {
                    i=2;
                    return null;
                }
                else
                i=posl;

        }catch (RecordStoreException ex) {
            i=2;
            prazdnaDB();
            return null;            }
        catch (IOException ex) {
             i=2;
            return null;
        }
        if(cteniDB(i))
        {
            return temp;
        }
        else
        {
            throw new Exception();              }
```

Metoda zapisDB

Tato metoda slouží k zápisu do databáze. Používá k tomu datový stream, který umožňuje zápis v původním formátu (double, String, …).

Veškerá data jsou uchovávána, takže časem by mohlo dojít k problému s pamětí, proto je v první části otestováno volné místo a pokud je málo paměti, tak dochází k mazání nejstarších dat v databázi (pětina databáze je smazána). Je mazáno od indexu dva (v prvním máme důležité informace). Poté již dochází k zápisu jednotlivých dat. Data jsou uložena do streamu a poté je celý zapsán pomocí metody `databaze.addRecord(data,0,data.length)`.

```
public int zapisDB(long cas, double tep, double saturace, double teplota,
String pad )
     {
       bout = new ByteArrayOutputStream();
       dout = new DataOutputStream(bout);
       try {
            if(databaze.getSizeAvailable()<1000)
             {
                 for(int i=2;i<(list.numRecords()/5);i++)
                 {
                     databaze.deleteRecord(list.nextRecordId());
                 }
             }
            dout.writeLong(cas);
            dout.writeDouble(tep);
            dout.writeDouble(saturace);
            dout.writeDouble(teplota);
            if(pad.equalsIgnoreCase("true"))
            {
                dout.writeUTF("ANO");
            }
            else
            {
                dout.writeUTF("NE");
            }
            byte[] data = bout.toByteArray();
            databaze.addRecord(data,0,data.length);
            dout.flush();
            dout.close();
            bout.close();
            return 1;
         } catch(RecordStoreException e){
            return 2;
         } catch(IOException e){
            return 3;
         }
      }
```

Použití

V prvním řádku dochází k inicializaci databáze. Poté je testováno, zda jsou v databázi nějaká neodeslaná data. Pokud jsou, dojde k otevření spojení a data jsou odeslána. Toto má na starosti třída ***Pripojeni***.

```
try {
      databaze=new Databaze();
      if((data=databaze.odeslano())!=null)
      {
          zaznam.zapis("Databaze obsahuje neodeslane
                       zaznamy.",typZapisu.Informativni);
          vlaknoUpload(false);
          if(data!=null){
              zaznam.zapis("Pokud odeslat
                           podruhe.",typZapisu.Informativni);
              vlaknoUpload(false);
              if(data!=null){
                  zaznam.zapis("Data ani na podruhe nebyly odeslany.
                               Upload olozen.",typZapisu.Chybovy);
              }
              else{
                  databaze.nastaveniOdeslano();
              }
          }
          else{
              databaze.nastaveniOdeslano();
          }


      }
```

Je zde i řešena situace, pokud by data nebyly odeslány (není připojení k internetu) – pokus data odeslat znovu a pokud opět nedojde k odeslání, je tento požadavek odložen na další plánovaný pokus (za hodinu). Nahrávání se nespustí také pokud není modul zaregistrován do sítě (byla by zbytečná spotřeba). Pokud jsou data odeslána, je volána metoda `nastaveniOdeslano()`, která zaregistruje odeslaná data (uloží nový index do indexu jedna v databázi). Metoda pro odeslání dat `vlaknoUpload(boolean prubeh)` je rozebrána v následující kapitole.

8.1.6 Třída Pripojeni

Tato třída slouží k odeslání dat do vzdálené databáze. Data odesílá pomocí http metody **POST**. Data jsou odesílána jako parametry a na servletu jsou překládána do SQL příkazů. Výstupní data jsou zapsána do výstupního streamu a odeslána. Při zapsání do databáze je vráceno příznak OK a je uloženo odeslání. V případě, že nedošlo k přijetí (nebylo přijato potvrzení), data jsou odeslána znovu při dalším pokusu. Ukázka:

```
public Pripojeni(Zaznam zaznam, String[][] data, Stav reset) throws
                                                              Exception {
        con = (HttpConnection)
        Connector.open("http://test.mediacalServlet.cz/");
        con.setRequestMethod(HttpConnection.POST);
        vstupniStream = con.openInputStream();
    }

    public void uploadDat() throws Exception
    {
        vystupniStream = con.openDataOutputStream();
        for (int i = 0; i < data[0].length; i++) {
              vystupniStream.writeUTF(data[0][i]);
        }
        if (con.getResponseCode() != HttpConnection.HTTP_OK)
        {
            up.zapis("Server nepotvrdil spravne prijeti dat.",
                     typZapisu.Chybovy);
            loop = false;
            throw new IllegalStateException();
        } else {
            loop = false;
            data = null;     // Data odeslany
        }
        vystupniStream.flush();
        vystupniStream.close();

    }
```

Dále je vidět samotné spuštění odeslání. Pokud vše proběhne jak má, data jsou potvrzena a je potvrzeno odeslání (zápis do indexu databáze). Data nejsou odeslána také v případě problémů s databází – pokud není otevřena, nedochází k odeslání (nebylo by co odeslat).

```
public void run() {
        while (loop) {
            try {
                uploadDat();
                up.zapis("Upload dat probehl uspesne",
                         typZapisu.Informativni);
                odeslano.setData(true);
            } catch (Exception ex) {
                up.zapis("Upload neprobehl z duvodu" + ex.toString(),
                         typZapisu.Chybovy);
                odeslano.setData(false);
```

8.1.7 Třída SPI

Tato třída slouží k příjmu a úpravě dat ze SPI. Pomocí metody `upravaDatSpi(int start,int konec)` dojde k úpravě dat (převod na jednotlivé parametry).

```
public void run() {
        prijataData.setData(false);
        zaznam.zapis("Vlakno na cteni dat a overeni dat z SPI
                    staruje.",typZapisu.Informativni);
        try{
            upravaDatSPI(0,7);
            prijataData.setData(true);
            zaznam.zapis("Data prijata v poradku.",typZapisu.Informativni);
        }catch(Exception ex){
            prijataData.setData(false);
            zaznam.zapis("Prijeti datSPI nebylo uspesne z
                        duvodu"+ex.toString(),typZapisu.Chybovy);
        }
```

Uložení dat je již ve třídě ***Jadro*** pomocí metody `vlaknoSPI(String dataSPI)`.

```
private synchronized void vlaknoSPI(String dataSPI)
    {
        if(SPIvlakno!=null)
        {
            SPIvlakno=null;
        }
        try{
                SPIvlakno=new Thread(new SPI(zaznam,dataSPI,reset,buffer));
                SPIvlakno.start();
                SPIvlakno.join(3000);
            }catch(Exception ex){
                zaznam.zapis("Prijeti dat se nepovedlo.
                            "+ex.toString(),typZapisu.Chybovy);
            }
        ulozeniDat(reset.getData());
    }
```

8.1.8 Třída Watchdog

Tato třída slouží ke kontrole programu a při problémech dochází k restartu zařízení. Navíc také kontroluje funkčnost zařízení – pokud nejsou přijata nová data do 15 minut, tak také dochází k restartu zařízení. Zařízení může fungovat, ale z nějakých důvodů nemusí přijímat data – jak bylo zmíněno, příjem funguje pouze při dostupných datech na vstupu.

```
public void start() {
        Watchdog.start(cas);
        loop = true;
        zaznam.zapis("WatchDog startuje", typZapisu.Informativni);
        vlakno.start();
    }

    public void stop() {
        loop = false;
        try {
            vlakno.join();
        } catch (InterruptedException ex) {
            vlakno.yield();
        }
    }
    public void run() {
        while (loop) {
            if (reset.getStav()) {
                zaznam.zapis("Watchdog ukoncuje aplikaci",
                             typZapisu.Chybovy);
                Watchdog.kick();
            }
            else{
                Watchdog.start(stop);
                Watchdog.start(cas);
                reset.setStav(true);
            }
```

Watchdog testuje proměnnou pomocí metody `getStav()` (viz knihovna ***Stav***) a pokud do definovaného času nedostane odpověď (tj. program nereaguje), dojde k ukončení programu. Toto vlákno běží s maximální prioritou. Každým průběhem tohoto vlákna se dekrementuje čítač, a pokud nejsou dostupná nová data do dané doby, dojde k ukončení.

Pokud jsou přijata nová data, je watchog restartován – pomocí metody `Watchdog.start(stop)` je watchdog vypnut (parametr `stop` je 0) a poté znovu spuštěn.

8.2 Software pro řízený blok

Tato část programu je napsána v programovacím jazyce C. Hlavním cílem této části je měření, prvotní vyhodnocování a odesílání dat. Jsou k dispozici dvě signalizační LED diody (*ALARM1 a ALARM2*). Ovládá pouze komunikaci, kdy řídícímu bloku oznamuje dostupnost dat pomocí DTR (resp. při komunikaci SPI pomocí pinu PD5). Běh programu je řízen a ovlivňován pomocí:

- **`INT0`** – *externí přerušení 0 – zrušení alarmu pomocí tlačítka, obsluhuje uživatel.*
- **`INT1`** – *externí přerušení 1 – pro změnu komunikace (UART -> SPI).*
- **`Reset`** – *reset zařízení při nestandardním chování.*
- **`DATA_OK`** – *signál připojen na PD5 – potvrzování příjmu dat. V případě, že není potvrzeno, jsou data odeslána znova. Při SPI komunikaci tento pin mění svojí funkci na oznamování řídícímu bloku dostupnost dat (uvažujeme DTR za nedostupný).*

Program je nastaven tak, aby i při dílčím selhání některého senzoru měření probíhalo dál a tento stav byl reportován. Proto se místy může kód zdát zbytečně složitý, ale je to nutné. Situace jde jednoduše vysvětlit na příkladu, kdy je tep nula – pacientovi se zastavilo srdce nebo jenom přestalo fungovat čidlo (příp. není spojení)? Proto je nutné určité parametry ověřovat i přesto, že to znamená zbytečné zdržení.

Program je postaven na hlavním programu (***`Hlavni`***), který využívá metody třídy ***`Data`*** – načítá, zpracovává data z čidel a výstupy jsou dva:

- **`Data v pořádku`** – pokračujeme v měření.
- **`Problém`** – generován alarm a zdroj alarmu je odeslán do řídícího bloku.

Třída ***`Data`*** využívá metod z jiných tříf, ostatních pomocných funkcí (UART, SPI,A/D).

Třída Hlavni

Na následující ukázce je vidět hlavní část kódu, kde dochází k prvotním inicializacím a poté již hlavní smyčka. Dochází k načítání dat z rozhraní UART (připojen čidlo tepu a saturace kyslíkem). Pokud je přijat celý rámec, je spuštěna metoda `kontrolaDat()` – ověří data a pokud jsou v daných limitech, data uloží do struktury. Ukázka kódu na další stránce.

```
int main(void)
    {

      initUART();
      initAD();
      initPort();
      /* Program */
      testSpojeni();

      while(1) {
                  if(nacteniRamce())
                  {
                        kontrolaDat();
                  }
                  if(startPrenosu)
                  {
                        dataOdeslat();
                        startPrenosu=0;
                  }
                  if(vyhodnoceni())
                  {
                              alarm();

                  }
                  if (nacteniTeploty())
                  {
                        dataOdeslat();
                  }
                  citac--;
                  if(!(citac))
                  {
                        zapis(alarm);
```

Na ukládání dat byla zvolena struktura (lepší z hlediska optimalizace než jednotlivé proměnné).

```
typedef struct{
      double tep;
      double saturace;
      double teplota;
      char pad;
      int prubeh;
}DATA;

DATA zaznam;
```

Z jednotlivých názvů proměnných, lze odvodit jejich účel – proměnná `prubeh` je inkrementována s každým uložením dat – data odesíláme po přijetí šedesáti vzorků. V posledním řádku je ukázka definice proměnné.

Metoda nacteniRamce()

Tato metoda čte data ze sériového rozhraní UART. Čtení je řízeno přerušením – tato metoda byla zvolena pro bezpečnost – metoda nečeká na data, ale v okamžiku přijetí dat, je generováno přerušení a přijatá data jsou uložena. Přerušení je obnoveno vyčtením dat z UDR. Takže program není blokován, pokud by data nebyla dostupné. Šedesát vzorků dat se počítá od těchto dat a poté jsou data odeslána. V případě nedostupnosti těchto dat se samozřejmě metoda `kontrolaDat()` nespouští a tím ani nedochází k počítání vzorků. Tato situace má dva „záložní“ mechanismy:

- **`Měření teploty`** – *zde také dochází k počítání vzorků a při překročení limitu (vyšší limit), dochází k odeslání dat*
- **`Čítač`** – *počítá průběhy hlavní smyčkou – měl by reagovat v nefunkčnosti obou dvou metod*

Data jsou tedy ukládána v obsluze přerušení. Metoda tedy provádí kontrolu, zda byl přijat celý rámec (testuje poslední znak >). Pokud jsou data přijata celá, je spuštěna metoda na kontrolu dat, jinak se pokračuje v činnosti.

metoda kontrolaDat()

```
void kontrolaDat()
{
	//...
	strncpy(temp,temp2,5);
	tep=atoi(temp2);

	if((tep>120)||(tep<50))
	{
		alarm2(0,temp2);
	}

	//...
	oxidace=atoi(temp2);
	if(oxidace<90)
	{
		alarm2(1,temp2);
	}

	ulozitData();

}
```

Na předcházející ukázce vidět část této metody. Dochází k rozdělení pole znaků a převedení na číslo (pomocí funkce atoi(char *s) – součást knihovny stdlib.h). Dál dochází k testu, zda nejsme mimo limit a poté k uložení. V případě překročení limitu, je generován alarm a tato data jsou odeslána do řídícího bloku. Pravidelně odesílaná data jsou průměrována, takže tam by se výkyv nemusel projevit (osamělý výkyv není ani v řídícím bloku brán v potaz). Na závěr je volána funkce pro uložení dat.

Metoda vyhonoceni()

Tato metoda slouží k načtení dat z A/D převodníku a jejich vyhodnocení. Ukázku lze vidět níže:

```
int vyhodnoceni()
{


     for(i=0;i<50;i++)          // 50 vzorku
       {
          znak[i]=prevod();
           prumer+=znak[i];
      }
      prumer=prumer/50;
      if (prumer>(0x300))
      {
            return 1;
      }
      else
            return 0;

}
```

Jak je vidět, je měřeno 50 vzorků a průměrováno. V případě překročení je generován alarm, ale data zatím nejsou předávána. Jak již bylo zmíněno, tento alarm není podruhé vyhodnocován. Dochází k „druhému“ vyhodnocení přímo zde. Je aplikován digitální filtr pro rozlišení stavů. Popsán je v kapitole Funkce.

V kódu jsou navíc přidány určitá zpoždění (možnost zareagovat uživatelem). Pokud dojde ke spuštění alarmu, je rozsvícena LED dioda *ALARM* a je provedeno 12 cyklů měření a pokud více jak 10 měření skončí nad limitními hodnotami je generován alarm (nízká úroveň na PD6).

```
void alarm(void)
{
	PORTB &= (0<<PB0);
	for(int i=0;i<12;i++)
	{
		suma=suma+vyhodnoceni();
		if(vlajka==1)
		{
			break;
		}
	}
	if((suma>10)||(vlajka==1))
	{
		PORTB |= _BV(PB0);
	}
	else
	{

		if(vlajka==0)
		{
			zaznam.pad='A';
			PORTD &= ~(_BV(PD2));
		}
```

Metoda nacteniTeploty()

Tato metoda slouží pro měření teploty (resp. přepočtu změny napětí při změně odporu na teplotu). Základem je funkce `vypocetTeploty()`. Jak již bylo popsáno výše, je použita Steinhart-Hartova aproximace.

V první fázi vypočteme napětí získané jako výsledek z A/D převodníku a rozlišením.

```
double vypocetTeploty(int vysledek)
{

	napeti=vysledek * rozliseni;
	 odpor = (7500 - napeti*refOdpor)/napeti;
	x=odpor/4700;
	SH = (A + B*log(x)+ C*(log(x)*(log(x))) + D*(log(x)*log(x)*(log(x))));
	teplota=(1/SH)-273.15;
	return teplota;
}
```

Dalším krokem je vypočíst odpor a ten poté dosadit to Steinhart-Hartovi rovnice. Jednotlivé parametry (A,B,C,D) jsou dosazeny z katalogu součástky. Výsledek je Kelvinech, proto dochází k přepočtu na stupně celsia (zlomek je tam, protože rovnice SH by měla být celá na mínus prvou). Samotná funkce již uloží hodnotu a inkrementuje počet záznamů a testuje, zda již nejsme přes limit (problém s načítáním dat z čidla tepu a saturace kyslíkem).

9. Funkce

Celé zařízení je složeno ze dvou hlavních bloků – řídící blok a řízený blok (měřící deska). Funkce celého zařízení lze vidět na vývojovém diagramu v Příloh. Tento vývojový diagram byl vytvořen v programu Diagram Designer[8] verze 1.25 (2012) – jedná o volně dostupný program bez jakéhokoliv omezení (freeware).

- **Řídící blok**
  - *GSM modul TC65.*
  - *Obslužný program napsán v Javě.*
  - *Má na starosti celý cyklus funkce, provádí vyhodnocení nestandardních stavů a dělá druhotné vyhodnocení dat.*
  - *Je zdrojem napájecího napětí pro měřící desku.*
  - *Program lze jednoduše upravit (i vzdáleně pomocí SMS).*
- **Řízený blok** *(měřící deska)*
  - *ATmega8, jednotlivá čidla, pomocné obvody.*
  - *Obsluhuje jednotlivé čidla.*
  - *Provádí prvotní vyhodnocení.*
  - *Odesílá naměřená data v daném formátu.*

Jednotlivé bloky jsou na sobě funkčně závislé. Pokud nebudeme uvažovat výpadek napájecího napětí, tak oba bloky mohou pracovat nezávisle na sobě, ale mohou zastávat pouze dílčí úlohy. Při výpadku měřící desky, řídící blok pracuje, ale nemá žádná data – pokud nejsou přijaty nová (validní) data do 15 minut, tak je generován reset od watchdogu. O této situaci je informován technik a uživatel – ten obdrží SMS o nefunkčnosti zařízení a o nutnosti provést kontrolu zapojení propojení (alespoň vizuální). Problém s komunikací s deskou je signalizován stavovou led diodou (zelenou), která za normálního stavu svítí. Tato LED dioda je přímo napojena na řídící obvod, proto funguje i při nefunkčnosti měřící desky.

Při nefunkčnosti řídícího modulu (opět při uvažování funkčnosti zdroje), je deska schopna měřit a vyhodnocovat data z čidel, je schopna generovat alarmy, ale pouze na úrovni této měřící desky – pomocí LED diod *ALARM1* a *ALARM2*. Data nejsou odeslána do řídícího modulu a tím pádem, není možná jakákoliv komunikace (záchranná služba, technik, …). Zařízení je v testovací fázi (jedná se o prototyp), proto je v současné době implementováno oznámení o funkčnosti – prozvonění uživatele při úspěšné inicializaci. Ve finálním výrobku by již tato funkce byla nežádoucí. Nefunkčnost lze nyní poznat podle nepotvrzení funkčnosti, ale k problémům může dojít během činnosti. Detekování tohoto stavu je možné např. podle trvalého nepotvrzování přijatých dat, případně z vnějšku při kontrole databáze na internetu podle dlouhodobější nedostupnosti dat. Jediné možné oznámení je pomocí LED diod na měřící desce – to by ale mohlo vést ke zmatení uživatele, že se jedná o alarm (i přesto, že generace obou alarmů najednou není možná – při generaci jednoho je pozastavena ostatní

[8] http://logicnet.dk/DiagramDesigner/

činnost a všechny prostředky jsou vyhrazeny řešení této situace). Při nefunkčnosti řídícího bloku může také přestat fungovat informační led dioda na GSM modulu.

9.1 Řídící blok

Při zapnutí zařízení dochází k dílčím inicializacím:

- **`Inicializace modemu`** – *AT příkazy, nastavení listeneru na AT příkazy, nastavení profilu, GPIO.*
- **`Inicializace sériového spojení`** – *sestavení spojení, nastavení listeneru, testovací příjem dat.*
- **`Inicializace databáze`** – *otevření, kontrola odeslání dat.*
- **`Odeslání dat`** – *test spojení, odeslání dat.*

Inicializace modemu

V první fázi dochází k inicializaci samotného modemu. Kritické nastavení je AT příkazů, jelikož bez nich je jakákoliv signalizace či ovládání některých částí nemožná. V případě problémů s tímto nastavení, je pouze restartováno zařízení. Další částí je inicializace GPIO sběrnice. Zde v případě problému je informován uživatel (kontrola zapojení) a technik – omezená komunikace je stále možná přes RS232 (bez potvrzování dat).

Pokud vše proběhne v pořádku (bez generace výjimky), je měřící deska uvedena do provozu (ATmega je do té doby držena v resetu).

Inicializace sériového spojení

V dalším kroku je testováno spojení. ATmega pošle předem daný rámec dat. Pokud řídící modul přijme celá data, je spojení ověřeno. V případě problému je přenos opakován – pokud je výsledek stejný, je přepnuto na záložní sběrnici a tato informace je poslána uživateli (opět kontrola zapojení) a technikovi. Z důvodu popsaných problémů se SPI, je původní spojení periodicky testováno (při každém odeslání dat – jednou za hodinu – je spouštěn testovací kód spojení, samozřejmě v případě, že byly problém se spojením). Můžeme uvážit situaci, kdy uživatel najde uvolněný kabel a zapojí zpět a tím můžeme přepnout zpět na toto spojení.

Kontrola toho spojení také probíhá pravidelným otevíráním a zavíráním – z důvodu nutnosti uzavřít spojení při nastavení GPRS profilu. Takže problémy jsou odhaleny i v průběhu (nejen z nedostupnosti dat).

Inicializace databáze

V tomto kroku dochází k otevření databáze – případné problémy jsou signalizovány generováním výjimek, které jsou řešeny v obslužném kódu. Zde již není informován uživatel, jelikož se jedná o chybu, kterou sám nemůže odstranit – jedná se o chybu modemu nebo JVM. Tento problém nijak neohrozí primární funkci – vyhodnocování krajních stavů – to stále funguje a stále lze generovat alarm. Pouze data nejsou ukládána do databáze.

Odeslání dat

Nastavení odeslání dat je každou hodinu (časování je plně v režii JVM – nemusí to být úplně přesné – v této činnosti na tom nezáleží). Pokud dojde k opakovaným problémům s odesláním dat, je informován technik. Může se jednat, že vzdálená databáze neběží (např. údržba), uživatel byl dlouhodoběji v oblasti bez signálu. Tato situace není řešena při prvním vzniku z výše popsaných důvodů.

Hlavní program

V hlavním programu je potvrzena funkčnost zařízení (prozvoněním uživatele) a zařízení je uspáno (mód číslo 9). Zařízení může být probuzeno několika způsoby:

- **`Aktivací listeneru`**
  - *RS232 listener (změna pinu DTR).*
  - *GPIO listener (změna pinu GPIO6).*
- **`AT příkazy`** *(při odesílání dat je modem převeden do plné funkčnosti).*
- **`Příjem SMS/Příchozí volání`**
- **`RTC alarm`**
- **`URC`**

Tento mód se probouzí pouze částečně – při generaci předchozích událostí, je modem probuzen jen dočasně – na vykonání obslužného kódu + čas navíc nastavený pomocí `AT^SCFG`. Obslužný kód listenerů je řešen spuštěním obslužného vlákna (doporučená metoda). A jelikož toto vlákno už běží mimo obslužný kód listeneru, tak je potřeba ponechat čas pro vykonání této činnosti. Samotné vlákno na čtení má vlastní limit 3 vteřiny, pak je ukončeno. Proto zde je limit nastaven na 5 vteřin (nastaven delší z důvodu možného ošetření nenačtení dat).

Každou hodinu je nastaveno odeslání dat do vzdálené databáze. Modem je vzbuzen do plné funkčnosti (potřeba k nastavení GPRS profilu) a data jsou odeslány. Přijmutí zpětného potvrzení a v případě kladného potvrzení, uložení příznaku odeslaných dat. Poté dojde k opětovnému otevření sériového spojení a k uspání zařízení. Jak již bylo výše popsáno, dochází zde i k opětovné kontrole spojení (pokud byl problém).

Toto je řídící blok. Je snaha o optimální nastavení hlášení problému a snahy nepřehltit uživatele a technika zprávami. Při nefunkčnosti měřící desky, může tento stav ohlásit. Uživatel tedy bude upozorněn na nefunkčnost a nebude na toto zařízení spoléhat. Při nefunkčnosti dílčích částí řídícího bloku je stále alespoň možná signalizace problému.

9.2 Řízený blok

Tento blok má na starosti obsluhu čidel, prvotní vyhodnocení dat a jejich odeslání. Po prvotní inicializaci řídícího bloku, dochází na inicializaci řízeného bloku:

- `Inicializace UART`
- `Inicializace A/D`
- `Inicializace portů`

Na začátku jsou poslána testovací data – správně přijatá data jsou signalizována vysokou úrovní na pinu PD5. V opačném případě jsou data odeslány znovu a program pokračuje (řešení je necháno na řídící blok).

Hlavní program

V hlavním programu dochází ke kontinuálnímu měření. Jsou měřeny 4 základní parametry:

- `Tep` – UART.
- `Saturace kyslíku` – UART (v jednom rámci s údajem o tepu).
- `Pád` – A/D převodník.
- `Teplota` – A/D převodník

Měření jde tedy rozdělit na dvě části.

Tep a saturace kyslíku

Data jsou posílána z čidla po UART v jednoduchém rámci:

```
<100.0,98.0>
```

První údaj je tep, druhý údaj saturace kyslíkem. Data jsou načtena a zkontrolována. Dochází k prvotnímu hodnocení – limity jsou nastaveny níže, při překročení je generován alarm a je odeslán. A je dále vyhodnocen v řídícím bloku – program v řídícím bloku se dá jednoduše přizpůsobit dle každého jedince (důvody popsány výše). Limity v měřící desce jsou nastaveny na standardní limity (dolní mez 60 tepů/minuta a horní 100 tepů/minut). Nyní je v řídícím bloku nastaven alarm o 10 níže resp. výše (50 tepů/minuta resp.110 tepů/minuta).

Problém může nastat při příjmu samých nul – může se jednat o zástavu srdce či čidlo odešlo. Bohužel nelze jednoduše rozlišit tyto stavy (nelze čekat na nějaké ověření od uživatele). Navíc pokud by čidlo odešlo, tak s vysokou pravděpodobností nebude posílat vůbec. Tento stav je signalizován do řídícího bloku. Čidlo tepu je vybaveno vnitřním senzorem na vložený prst – neměří, pokud uživatel vyndá prst.

Data jsou přijímána a vyhodnocována každou sekundu. Jsou dělány minutové průměry a ty jsou odesílány.

Pád a teplota

Napětí na výstupu akcelerometru je přivedeno na A/D převodník. Je průměrováno 50 vzorků – pád lidského těla má setrvačnost. Poté je načtena teplota – napětí na odporu, který je v sérii s termistorem, je přivedeno na další vstup mikrokontroléru.

Pokud dojde k překročení hranice pro zrychlení v ose z, je spuštěna funkce alarm, která ověří, zda se opravdu jedná o alarm. Princip je jednoduchý – pokud člověk upadne, další výrazné dlouhodobé zrychlení v původním směru pádu nevzniká. Proto je provedeno 12 cyklů měření, a pokud ve více jak dvou třetinách těchto měření vyjde zrychlení větší než limit, je alarm zrušen. Tato situace může značit, že uživatel se rozběhl, poskakuje, … Těchto 12 cyklů trvá zhruba 3 sekundy. V opačné situaci je čekáno dalších 5 vteřin (uživatel padl, ale nic se mu nestalo) na zrušení alarmu pomocí tlačítka.

Poté je generován alarm. Pouze v tomto případě není tento alarm vyhodnocován v řídícím bloku – alarm je signalizován změnou na nízkou úroveň na pinu PD4. Nepotřebujeme přenášet informaci o hodnotě překročení limitu.

Teplota je brána pouze jako doplňková hodnota. Při život ohrožujících teplotách, bude jistě tep (saturace kyslíkem) mimo limit. Teplota negeneruje alarm. Řídící blok vyhodnocuje teplotu a v případě nárůstu o 1 stupeň (do 39°C resp. pod 36°C) informuje uživatele, nad 39°C resp. pod 36°C upozorňuje i doktora po půl stupních (ale vždy jednou za půl stupně). Měření teploty probíhá vždy po cyklu měření z akcelerometru. Není nutné kontinuálně měřit teplotu, jelikož teplota nevzrůstá skokově.

Generace alarmu

Po vyhodnocení alarmu a poslání dat, pokračuje měření dál. Tato data jsou vyhodnocena řídícím modulem a ten rozhodne o dalším řešení. Není zde žádná zpětná vazba – ani při generaci alarmu z řídícího bloku. Data jsou měřena dále. Další data jsou důležitá – můžeme monitorovat, co se děje. Může to být důležitá informace pro lékaře, co se dělo s uživatelem a jeho životními funkcemi. Data jsou proto dále měřena a ukládána do databáze. V řídícím bloku je zajištěno, že další alarm už nebude generován.

10. Vyhodnocení

10.1 Alternativy na trhu

Při návrhu tohoto tématu jsem zkoumal, zda je nějaké podobné zařízení dostupné. Zjistil jsem, že ne. Běžně komerčně dostupné jsou pouze zařízení s dílčími funkcemi (např. hodinky s monitorováním tepu a saturace), ale žádné zařízení se všemi funkcemi. A navíc všechna tato zařízení mají pouze monitorovací funkci – žádné z nich není určeno pro funkci zachraňování životů. Jedná se většinou o různé sportovní pomůcky (spalování tuků je ideální jen do jisté tepové hranice, pak již je kontraproduktivní), příp. zařízení typu Holter – zařízení používané pro kontinuální zaznamenávání dat (tepu, tlaku,…) používané na dlouhodobější měření (typicky 24 hodin). Toto zařízení data pouze zaznamenává a poté pacient s tímto zařízením musí k lékaři, který si data ze zařízení musí nahrát a vyhodnotit.

Začíná se rozvíjet jistá produktová škála od společnosti Cinterion nazvaná *mHealth,* která podle slov autorů, chce využít prázdného místa na trhu s lékařským vybavením [30]. Sami autoři konstatují, že je tu velká mezera, kdy technologie je dostupná pro tyto účely. Zařízení této společnosti by se daly rozdělit do dvou skupin:

- **Diagnostikující zařízení**
- **Monitorovací zařízení**

V prvním případě se jedná o zařízení, které monitoruje několik parametrů a na základě nich vyhodnocuje zdravotní stav pacienta (určení je spíše pro méně závažné situace).

V druhém případě se jedná o velmi podobné zařízení jako v této práci. V současné situaci je založeno na monitorování dat a bezdrátovém odesílání dat na speciální portál, kde autorizovaný uživatel může sledovat průběžná data. Navíc může z tohoto portálu odeslat zprávu (např. zda je osoba v pořádku) – tato zpráva se zobrazí na displeji zařízení pacienta a ten jedním kliknutím může odpovědět. Tato skupina obsahuje i speciální přídavné možnosti jako monitorování cukru (pro lidi s cukrovkou) ze vzorku krve, možnost automatického připomenutí léků atd.

Tyto zařízení jsou prozatím ve fázi „ukázkových prototypů“, kdy jejich cena je velmi vysoká.

10.2 Realizace

Jak již bylo napsáno, v době návrhu tohoto tématu nebylo dostupné na trhu podobně komplexní zařízení. Rozhodl jsem se pro tento návrh funkční prototypu z několika důvodů – hlavní důvodem bylo možnost postavit smysluplné zařízení, které by při rozšíření mohlo zachraňovat lidské životy. Nechtěl jsem pouze vyrobit „něco“, co by fungovalo, ale nemělo nějaký smysl a využití. Toto zařízení je založeno na aktuální potřebě a má svůj cíl. Jelikož toto zařízení je prototyp, setkal jsem se během vývoje s několika problémy, které jsem musel

řešit. Navíc se jedná zařízení, které „nesmí“ selhat, jelikož uživatel (pacient) na něj spoléhá (nejedná o zařízení typu bankomat, kdy nefunkčnost maximálně klienta rozzlobí, ale neublíží mu). V této situaci nefunkčnost může mít velký dopad. I z tohoto důvodu bylo nutné na vývoj zařízení přistupovat z různých úhlů a snažit se předejít problémům – ať už se jedná o rozpojení komunikačního kabelu, až po selhání dílčích částí (nikdy nelze zaručit, že některé zařízení/součástka bude fungovat celou dobu).

Jsou zde kritické části, u kterých jde jen těžko zajistit nápravu – např. napájecí zdroj (baterie) – pokud dojde k výpadku, zařízení přestává fungovat – samozřejmě by se dalo argumentovat záložním napájením, ale to by šlo i u selhání komunikačního zařízení a nakonec bychom se dostali do situace, kdy zařízení by uživatel musel nosit v batohu na zádech. Cílem bylo zařízení udržet mobilní s maximalizací možných ochranných (záložních) možností.

Důraz byl kladen na alespoň minimální funkci při selhání dílčích částí, proto se někdy může zdát, že místy jsou zvolené postupy nadbytečné, ale je to nutné pro zajištění funkčnosti. Například u měřící desky – jde říci, že odesílání (počítání vzorků) by stačilo podle čidla tepu, ale co když toto čidlo přestane fungovat, příp. uvolní se spojení? Proto je zde záložní řešení odesílání dle dat z teploměru. A i při tomto selhání je alespoň odeslán údaj o nedostupnosti dat.

Na druhou stranu bylo důležité ošetřit falešné alarmy – např. při výpadku měření tepu by byla odeslána nula, což by bylo vyhodnoceno jako zástava krevního oběhu a volána záchranná služba kvůli uvolněnému drátku. Proto je implementováno několik testovacích režimů pro zjištění, zda jednotlivá čidla pracují, jak mají. Data jsou podrobena dvojímu vyhodnocení. Řídící blok je navržen pro jednoduchou úpravu softwaru pro snadnou úpravu „na míru.“ Jde jednoduše upravit meze podle požadavků a lze i nakonfigurovat vyhodnocované parametry.

V rámci této práce byla řešena jak hardwarová i softwarová část řešení. Byla snaha tyto dvě části skloubit dohromady k zajištění bezchybné funkčnosti.

Během této práce jsem se setkal s několika problémy. Původní čidlo pro měření tepu a saturace kyslíkem bylo SMITH PulseOximeter, které bylo 4 roky po certifikaci (pravidelné kontroly pro posouzení certifikace pro lékařské použití – tzv. „záruka funkčnosti“). Nastal problém – přerušení sběrnice od čidla uvnitř zařízení, který se již nepovedl opravit. Dalším čidlem bylo PulseOximetr Lxx5 od společnosti AliExpress, které po jednom dni přestalo fungovat. Velkým problémem byla komunikace s jednotlivými firmami – většinou nemají zastoupení v České republice a na emailové dotazy neodpovídaly. Byla snaha sehnat jiná čidla (příp. OEM řešení), ale bez jakékoliv odezvy.

GSM modul TC65

Tato část se zabývá problémy v průběhu vývoje. Předem bych chtěl poznamenat, že byl použit dodávaný software od výrobce s doporučeným nastavením od výrobce. Většina obrázků zde, jsou z programu Netbeans 5.5.1, ale ty samé situace lze reprodukovat i v druhém doporučeném programu Eclipse. Propojení bylo přímým kabelem bez převodníků (USB/RS232).

Tento modul netrpí jen výše popsaným problémem – nedotažený soubor nástrojů pro vývoj aplikace (SDK), ale také tato verze (Terminal – v krabičce) je ochuzena o některé vlastnosti – např. USB. Deska obsahuje USB řadič, ale není vyveden konektor. To je daň za poměrně robustní ochranu (využití lze najít v průmyslových aplikacích, kdy zařízení může být vystavováno nevhodným podmínkám – prach, nárazy, …).

Během této práce jsem narazil hned na několik problémů (chyb) tohoto modulu (či spíše vývojového prostředí potažmo komunikace mezi vývojovým prostředím a modulem). Z počátku se zdály jednotlivé chyby spíše náhodné, ale při intenzivnější práci se zařízením jsem zjistil, že chyby se opakují.

První větší skupinou problémů je komunikace mezi modulem a vývojovým softwarem (či špatné vyhodnocování stavů). První příkladem je hlášení o nemožnosti spuštění programu uvnitř modulu. Jednoduchým testem (posílání textu po RS232) bylo ověřeno, že i přes toto hlášení byl program spuštěn. Při testech byl vypnutý autostart, takže program musel být spuštěn tímto. Tento problém lze hodnotit jako méně závažný – po zjištění nepravdivosti hlášení lze toto hlášení ignorovat.

```
>>> Starting Download of Jar and Jad file... <<<

Downloading "C:\Documents and Settings\Tomas\Diplomka_0.9.1\dist\nbrun43403\Diplomka_0.9.1.jad"...
        ...finished

Downloading "C:\Documents and Settings\Tomas\Diplomka_0.9.1\dist\nbrun43403\Diplomka_0.9.1.jar"...
        ...finished

Start Java program inside the module...


No start of Java program is possible. Start is aborted!
BUILD SUCCESSFUL (total time: 15 seconds)
```

Obr. 11: Start programu

Dalším problémem je „zablokování“ komunikační linky, kdy někdy při stahování programu do modulu, je vrácena chyba:

```
"C:\Documents and Settings\Tomas\Application Data\Siemens\SMTK\TC65_R3\WM_Debug_config.ini"

COM port used for "emulator session": COM1

Used baud rate for the module "115200 baud"...

Unable to use selected baud rate for the module.
Please check, if "COM1" is set to "115200 baud" or is used from another device!
Please use "at+ipr=115200" to set correct baud rate!
Communication problems to the module.
Please check the COM port and the connections!
No initialization for module debugging is possible!
BUILD SUCCESSFUL (total time: 8 seconds)
```

Obr. 12: Selhání komunikace

Jak je vidět z předešlého obrázku, jsou hlášeny komunikační problémy. Jako možná příčina je špatné nastavení COM portu nebo použití COM portu jinou aplikací. Tato chyba jde někdy reprodukovat – opakovaným spouštěním nahrávání programu dostáváme stejnou chybu. Někdy druhý pokus již projde v pořádku. V prvním případě pomáhá uvolnění komunikační linky postupem, kdy pouze otevřeme spojení přes hypeterminál a hned zavřeme a vše funguje jak má.

Další skupina problémů souvisí s obslužným programem pro systém Windows – MES Window tool (Module Exchange Suite – program pro přístup k zařízení). Zde se objevují dva problémy:

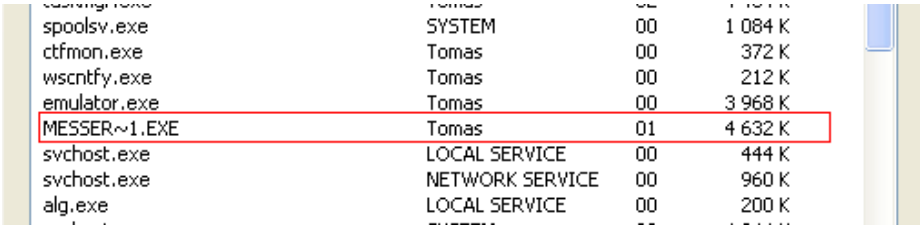
- Kopírování souborů – *občasné problémy s kopírováním souborů ze zařízení (většinou vyřešil restart).*
- Řádné ukončení – *program se občas řádně neukončil a tím blokoval zápis do modulu.*

Zastavíme se na chvíli u druhého problému. S tímto problémem jsem se setkával poměrně dost často, jelikož zařízení zapisuje svůj průběh do souboru, který je uložen v paměti přístroje. Jediná možnost připojení je program MES od výrobce. Při vývoji software byly pro mě důležité jednotlivé logy, a proto jsem často přistupoval do paměti a často jsem setkal s tímto výpisem:

```
Downloading "C:\Documents and Settings\Tomas\Diplomka_0.9.1\dist\nbrun29629\Diplomka_0.9.1.jar"...
        ...finished

Start Java program inside the module...

Please start module and emulator again!


No start of Java program is possible. Start is aborted!

Please close any other running Flash File access program (e.g. MES Window tool or MES commandline tool)!
BUILD SUCCESSFUL (total time: 16 seconds)
```

Obr. 13: Přístup

Přestože program MES byl řádně uzavřen, byl odepřen přístup do paměti. Jednoduchou kontrolou aplikace Správce úloh, bylo zjištěno, že program MES zůstal v paměti:



Obr. 14: Program MES - ukončení

Program lze ukončit jako nefunkční program, ale většinou má za následek restart programu *explorer.exe*.

Třetím problémem je pravděpodobně chyba syntaktické analýzy („parser“). Problém byl objeven se znakem ^ („stříška“, v angličtině *caret*). Všechny ostatní znaky fungují naprosto normálně, ale s tímto znakem je problém. Část problému lze odstranit zapsáním znaku v hexadecimálním kódu, ale tím není vyhráno. Pro objasnění přiložen obrázek:

```
AT^SCFG="AutoExec",1,1,2,5,"AT^SJNET?","000:00:10"
^SCFG: "AutoExec","0","0","0","0",""
^SCFG: "AutoExec","0","1","0","0","","000:00:00","086:28:15"
^SCFG: "AutoExec","0","1","1","0","","000:00:00","086:28:15"
^SCFG: "AutoExec","1","1","2","5","AT^SJNET?","000:00:10","000:00:09"

OK

^SCFG: "AutoExec","1","1","2","5","AT■SJNET?"

ERROR
```

Obr. 15: Problém syntaktické analýzy

V první řádce je poslán AT příkaz, který naplánuje vykonání zadaného AT příkazu v daných periodách. Na čtvrté řádce lze vidět, že tento příkaz je potvrzen (znak ^ je interpretován dobře) a je zobrazen čas do příštího vykonání. Následuje potvrzení, že vše je správně. Bohužel vykonání již neproběhne v pořádku – lze vidět, že znak ^ byl změněn na jiný. Mezi uložením a vykonáním dojde ke špatné interpretaci tohoto znaku, jelikož po zadání příkazu je vše v pořádku, ale při vykonání už je problém. Příkaz k vykonání je kontrolován, zda je správně - nelze tedy spustit nefunkční případ.

Problémy či spíše špatná implementace ovládání rozhraní SPI, byly již nastíněny v rámci celé práce. Je zde pouze jeden příkaz na odeslání dat (příp. s malou modifikací o požadavek příjmu dat).

S popsanými problémy jsem se setkal během této práce. Na různých technických fórech lze nalézt daleko více odhalených chyb, které ale nemohu potvrdit (nesetkal jsem se s nimi či jsou mimo způsob používání). Předposlední problém ovlivnil vývoj programu – bylo

plánováno použít tento příkaz pro spouštění odeslání dat do databáze, ale z důvodu nefunkčnosti byla metoda odeslání předělána na časování samotného vlákna pro odeslání. Ostatní problémy se vyskytly při komunikaci (nahrávání programu) a tudíž nemají vliv na funkčnost.

Problém s audio zařízením – bylo v plánu přidat na měřící desku zvukové zařízení, které by oznamovalo problematické stavy uživateli, a tím by nemusel jen sledovat na generované alarmy pomocí LED diod. Tento způsob by byl daleko lepší z hlediska rychlosti reakce uživatele. Bohužel audio zařízení se nepodařilo zprovoznit.

Na závěr bych zmínil problém funkčnosti pod Windows 7, ale to nemůžeme považovat za chybu, jelikož je vývojové prostředí určeno pod Windows XP. Na stejném principu je i nefunkčnost v novějších verzích Javy – kde ale tento problém je znát – nemožnost použít novější metody /které přidávají nové funkce či jsou lépe optimalizované).

Další postup

Jedná se o prototyp - jedno z prvních zařízení tohoto typu na trhu, tudíž nebyla možnost se inspirovat již hotovým řešením. Jak již bylo napsáno, prvním volbou bylo zařízení Sun Spot, které má novější implementaci Javy, je daleko menší, má vlastní baterku, je osazeno Wi-Fi, …), ale nebylo k dostání. Jako alternativa byl zvolen tento GSM modul. Pro další vývoj by bylo optimální přejít na jiné zařízení (např. na zmíněný Sun Spot), který také běží na bázi Javy, takže by stačila pouze úprava stávajícího kódu a bylo by hotovo (v podstatě nahradit implementaci od Siemensu) – tím by minimálně odpadl problém se špatnou implementací Javy (jelikož zařízení Sun Spot je od stejné firmy, která stojí za vývojem Javy).

S přechodem na jiné zařízení by také šlo uvažovat přidání dalšího komunikačního prostředku a to např. Wi-Fi. Signál by se dal pokrýt např. byt, zahrada atd. Data přenášet pomocí toho spojení a poté např. pevným připojením přenášet do databáze. Navíc by šel napsat jednoduchý program pro zobrazování dat. Pokud by tedy byla potřeba kontinuálně zobrazovat data, nebyl by problém.

Zařízení bylo navrženo s cílem zachraňovat lidské životy. Samozřejmě by zařízení šlo rozšířit o další prvky – např. o prvky zpříjemňující použití (LCD displej s údaji, možnost nastavení,…), ale toto zařízení bylo vyrobeno jako funkční vzorek k jednomu účelu. Cílem nebylo vyvinout multifunkční zařízení. Cílem nebylo vyvinout zařízení pro sportovce, kteří si potřebují monitorovat tep pro dosažení maximálních výsledků, ani zařízení pro vyšetření pacienta, kdy nás zajímají okamžité výsledky, cílem bylo zařízení, které řeší život ohrožující stavy, příp. hraniční hodnoty.

Použitý napájecí zdroj (baterie + step-up měnič) je jednoduchým ukázkovým provedením pro testování. Pro další implementaci by bylo vhodné přejít na jiný zdroj (např.

použít stejný typ baterií jako pro mobilní telefony). Další případné rozšíření v této části by bylo možné provést úplné spojení s mobilním telefonem, který by tím nahradil GSM modul. Dala by se napsat další aplikace, která by umožňovala zobrazování měřených hodnot a jejich následné odesílání. Toto řešení by mělo i výhody integrace dvou zařízení do jednoho – uživatel by měl vše na jednom místě (měřená data, informace o funkčnosti zařízení příp. problémy).

Další věc spojená s napájením, je oddělení napájení pro desku od GSM modulu. Kdy by šlo přidělat zvukové zařízení při rozpojení GPIO konektoru a tím by uživatel mohl být upozorňován na problém.

V kapitole Akcelerometr bylo popsáno připojení všech tří os, přestože používáme pouze jednu. Akcelerometr by šel použít pro kontrolu pohybu pacienta – někdy je důležité, aby pacient dodržoval přiměřený pohyb ke zlepšení prokrvení. Dalším krokem by bylo doplnění zařízení o tuto funkci – šlo by pouze o úpravu software. Situace by byla opačná než teď – nyní nás zajímají krátké skokové změny – v případě pohybu by šlo o monotónnější úseky.

8. Závěr

První část této práce je zaměřena na teoretický úvod do problematiky. Myslím si, že není na škodu uvést základy použitých technologií. Navíc jedná z části mé práce se má zabývat bezpečností napsané aplikace v JavaME, proto část o tomto programovacím jazyce je obsáhlejší, kde jsou zmíněny základní principy a základy bezpečnosti. Druhá část je zaměřena na praktickou realizaci. Nejprve dochází k popisu jednotlivých bloků (příp. důležitých součástek), dále dochází k popisu softwaru – software pro řídící modul je rozebrán poměrně podrobně a to, protože se jedná o stěžejní část – část, která rozhoduje o řešení dílčích situací a má na starosti celý běh.

Cílem práce bylo prozkoumat využití embedded modulů pro monitoring životních funkcí pacienta. V původním návrhu řídící blok byl postaven na zařízení Sun Spot, ale nakonec nebylo dostupné, proto byla zvolena alternativa v podobě GSM modulu TC65. Tento modul se více hodí pro industriální použití, proto celý návrh musel být upraven.

V průběhu návrhu se systém rozrůstal – bylo nutné přidávat další části (ať už z důvodu změny řídícího bloku či zvýšení zabezpečení). Primárním komunikačním kanálem bylo zvoleno rozhraní RS232 (lepší implementace) a jako záložní SPI. Porovnávat zde rychlosti jednotlivých rozhraní nemá cenu, jelikož přenášíme data o malém objemu. Na měřící desce jsou 3 LED diody pro signalizaci stavů (alarm pád, alarm tep/saturace kyslíkem a signalizace stavu měřící desky) – první dvě jsou řízeny měřící deskou, třetí je řízena z GSM modulu – pro signalizaci problému s měřící deskou. Další LED dioda je v napájecím obvodě – pro signalizaci slabé baterie. Poslední LED dioda je stavová dioda GSM modulu – registrace do sítě, režim spánku, atd.

Akcelerometr a teploměr jsou napevno připojeny, ale čidlo tepu a saturace kyslíkem lze v případě potřeby vyměnit. Je připojeno na šroubovací konektor, takže lze jednoduše vyjmout a nahradit jiným. V tom případě by byla pravděpodobně nutná úprava softwaru (data musí mít předem stanovený formát).

Bylo zvoleno „dvojí vyhodnocování“, kdy v prvním kroku (měřící deska) dochází k vyhodnocování dat a meze jsou nastaveny níže a proto jsou reportovány i stavy, které samy o sobě nemusejí znamenat nic vážného. Hranice pro tep byly nastaveny dle doporučení lékařů – nad 100 a pod 60 tepů za minutu. Druhé vyhodnocení spočívá v modulu TC65, který má „finální slovo.“ Toto je výhodné i pro možnou adaptabilitu – je možné přístroj přizpůsobit více pacientům (u některých pacientů může tep 120 znamenat problém a u některých ne – proto se dá upravit na míru, tyto rozhodovací úrovně se můžou měnit a jednoduše pacient ani nemusí docházet na servisní místo kvůli přeprogramování – vše se dá udělat vzdáleně pomocí SMS – služba OTAP). Jak již bylo zmíněno, zařízení není postaveno jako víceúčelové zařízení s přívětivým uživatelským rozhraním. Navíc by to u tohoto zařízení mohlo být kontraproduktivní. Zařízení je především určeno starším lidem, kde by mohlo dojít ke špatnému nastavení – pacient si omylem může přenastavit jeden či více atributů na nereálnou hodnotu, čímž se zařízení stává v podstatě nefunkčním. Změny nastavení zařízení

je možné, jak již bylo zmíněno, provést na dálku pomocí služby OTAP, kdy je stažen nový program z webového serveru. Samozřejmě může se nabízet otázka, že při nefunkčnosti nebude technik u zařízení, ale tato otázka je smazána s tím, že program bude testován na tom samém zařízení (na tom samém typu). Situace není jako např. s operačním systémem Android, který je pro mnoho typů mobilních telefonů a proto se u některých typů může vyskytnout nefunkčnost. Zde je případ jiný.

Zařízení se snaží diagnostikovat celý chod a zjištěné závažné problémy hlásit pacientovi/technikovi. Každá součástka má životnost, takže může dojít k nefunkčnosti a to je řešeno určitou signalizací (musíme vzít v potaz, pokud neodejde zdroj – v tomto případě nelze vzniklou situaci, jakkoliv signalizovat, ale zařízení nepůjde zapnout, „nic nikde nebude svítit“, z čehož může uživatel jednoduše vyvodit, že je zde nějaký problém a bude kontaktovat technika).

Zařízení bylo provedeno jako prototyp. Pokud vezmeme v potaz doporučená vylepšení (alternativní postupy), je poměrně široká možnost dalšího vývoje a následného využití.

Bibliografie

1. **Heath, Steve.** *Multimedia and communications technology.* Waltham : Focal press, 1999.

2. AT commands. [Online] [Citace: 5. 9 2011.] http://nemesis.lonestar.org/reference/telecom/modems/at/summary-at.html.

3. ATCommand Ref. [Online] [Citace: 6. 9 2011.] http://remotesatellite.com/support/iridium/ISU-AT-Command-Ref-v2.pdf.

4. ATCommand Guide. [Online] [Citace: 6. 9 2011.] http://linmodems.technion.ac.il/pctel-linux/Pctel.ATCommand.Guide.6.23.00.pdf.

5. Datasheet TC65 ATCommands. *tc65_at_command_set.pdf.*

6. Datasheet TC65 Terminal Hardware Description. *TC65_Terminal_Hardware_Description.pdf.*

7. **Zakhour, Sharon a kolektiv.** *Java 6 Výukový kurz.* Brno : Computer press, a.s., 2007.

8. Oracle. [Online] [Citace: 9. 10 2011.] http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html.

9. Java Language Enviroment. [Online] [Citace: 12. 11 2011.] http://java.sun.com/docs/white/langenv/Intro.doc2.html.

10. **Mahmoud, Qusay H.** *Naučte se Java 2 Micro Edition.* Praha : Grada, 2002.

11. **Sing Li, Jonathan Knudsen.** *Beginning J2ME: from novice to professional.* New York : Apress, 2005.

12. **Rischpater, Ray.** *Beginning Java ME Platform.* Apress : New York, 2001.

13. KVM. [Online] [Citace: 20. 09 2011.] http://java.sun.com/products/cldc/wp/.

14. CLDC. [Online] [Citace: 28. 9 2011.] http://jcp.org/en/jsr/proposalDetails?id=139.

15. JSR58. [Online] [Citace: 6. 11 2011.] http://jcp.org/en/jsr/detail?id=58.

16. JSR228. [Online] [Citace: 11. 10 2011.] http://jcp.org/en/jsr/proposalDetails?id=228.

17. IMlet. [Online] [Citace: 11. 10 2011.] http://developers.sun.com/mobility/imp/impstart/.

18. **Mukhar, Kevin.** *Beginnig Java EE 5.* New York : Apress, 2005.

19. Akcelerometr. [Online] [Citace: 11. 12 2011.] http://www.micro.feld.cvut.cz/home/X34SES/prednasky/08%20Akcelerometry.pdf.

20. Oximetry principles. [Online] [Citace: 28. 04 2012.] www.oximetry.org/pulseox/principles.htm.

21. Stein-Hart equation. [Online] [Citace: 30. 01 2012.] http://www.ilxlightwave.cn/appnotes/thermistor_calibration_steinhart-hart_equation.pdf.

22. Datasheet MMA7260QT. *MMA7260Q.pdf.*

23. Datasheet NTC640. *NTC_Thermistors_Vishay.pdf.*

24. Datasheet ATmega8. *doc2486.pdf.*

25. AVR Hardware Design Considerations. *doc2521.pdf.*

26. Datasheet MAX232. *max232.pdf.*

27. Datasheet LF33CV. *LF33CV.pdf.*

28. **Hammerbauer, Jiří.** *Elektronické napájecí zdroje a akumulátory.* Plzeň : ZČU, 1998.

29. Datasheet LM2577. *lm25577.pdf.*

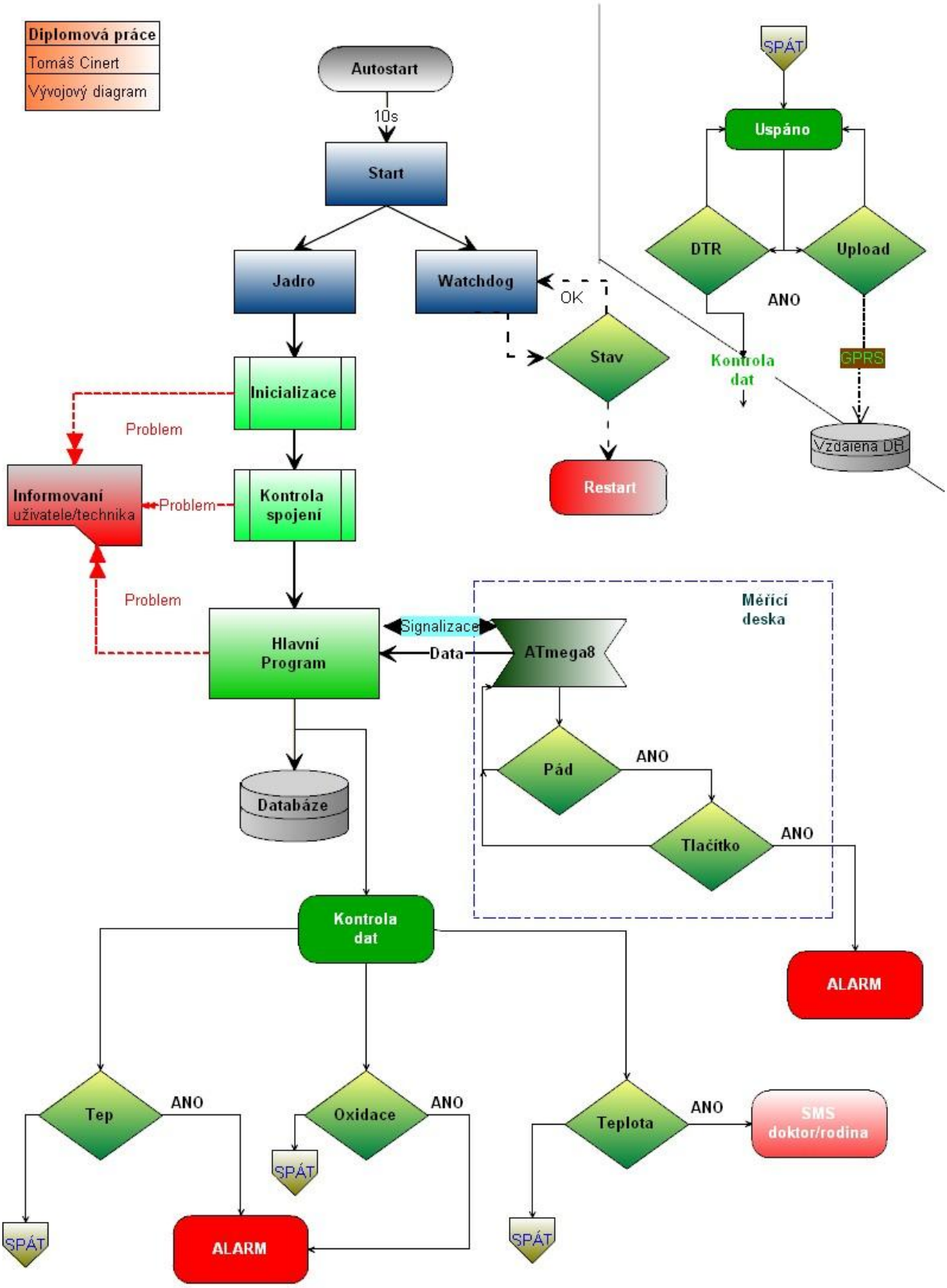
30. Cinterion mHealth. [Online] [Citace: 27. 04 2012.] http://www.cinterion.com/m2m-world/m2m-markets/mHealth.html.

Seznam obrázků

Seznam tabulek a grafů

Přílohy

Příloha A – Vývojový diagram



Příloha B - Schéma